SELECTIVE ALGORITHMS FOR LARGE-SCALE CLASSIFICATION AND STRUCTURED LEARNING

BY

KAI-WEI CHANG

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Doctoral Committee:

Professor Dan Roth, Chair
Professor David Forsyth
Professor ChengXiang Zhai
Doctor John Platt, Google

Abstract

The desired output in many machine learning tasks is a structured object, such as tree, clustering, or sequence. Learning accurate prediction models for such problems requires training on large amounts of data, making use of expressive features and performing global inference that simultaneously assigns values to all interrelated nodes in the structure. All these contribute to significant scalability problems. In this thesis, we describe a collection of results that address several aspects of these problems – by carefully selecting and caching samples, structures, or latent items.

Our results lead to efficient learning algorithms for large-scale binary classification models, structured prediction models and for online clustering models which, in turn, support reduction in problem size, improvements in training and evaluation speed and improved performance. We have used our algorithms to learn expressive models from large amounts of annotated data and achieve state-of-the art performance on several natural language processing tasks.

Publication Notes

Parts of the work in this thesis have appeared in the following publications:

- K.-W. Chang and D. Roth. Selective block minimization for faster convergence of limited memory large-scale linear models. In *Proc. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2011.
- K.-W. Chang, R. Samdani, A. Rozovskaya, N. Rizzolo, M. Sammons, and D. Roth. Inference protocols for coreference resolution. In *Proc. of the Conference on Computational Natural Language Learning (CoNLL) Shared Task*, 2011.
- K.-W. Chang, R. Samdani, A. Rozovskaya, M. Sammons, and D. Roth. Illinois-coref: The UI system in the conll-2012 shared task. In *Proc. of the Conference on Computational Natural Language Learning (CoNLL) Shared Task*, 2012.
- K.-W. Chang, R. Samdani, and D. Roth. A constrained latent variable model for coreference resolution. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013.
- K.-W. Chang, V. Srikumar, and D. Roth. Multi-core structural svm training. In *Proc. of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 2013.
- K.-W. Chang, S. Upadhyay, G. Kundu, and D. Roth. Structural learning with amortized inference. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2015.

Other relevant publications by the author:

- K.-W. Chang, A. Krishnamurthy, A. Agarwal, H. Daume III, J. Langford. Learning to Search Better Than Your Teacher. Arxiv:1502.02206 (preprint), 2015.
- K.-W. Chang, B. Deka, W.-M. W. Hwu, and D. Roth. Efficient pattern-based time series classification on GPU. In *Proc. of the IEEE International Conference on Data Mining (ICDM)*, 2012.
- R. Samdani, K.-W. Chang, and D. Roth. A discriminative latent variable model for online clustering. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2014.
- K.-W. Chang, W.-t. Yih, B. Yang, and C. Meek. Typed tensor decomposition of knowledge bases for relation extraction. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, 2014.
- K.-W. Chang, S. Sundararajan, S. S. Keerthi, and S. Sellamanickam. Tractable semi-supervised learning of complex structured prediction models. In *Proceedings of the European Conference on Machine Learning (ECML)*, 2013.
- H.-F. Yu, C.-J. Hsieh, K.-W. Chang, and C.-J. Lin. Large linear classification when data cannot fit in memory. In *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD)*, 2010.
- C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2008.

To my family and my love.

Acknowledgments

I would like to express my sincere thanks to my advisor, Prof. Dan Roth, for his guidance and tremendous support over the past five years. Dan is not only a remarkable scholar but also a passionate educator. His foresight in research, his dedication to mentoring, and his open-minded personality have greatly influenced me. He has demonstrated what it means to be a great advisor.

I was fortunate to have David Forsyth, ChengXiang Zhai, and John Platt on my dissertation committee, from which I got valuable and constructive comments. I am lucky to have worked with and to have been mentored by John Langford, Hal Daum'e III, Alekh Agrawal, Scott Yih, Chris Meek, Sathiya Keerthi, and Sundararajan Sellamanickam during my internship. I benefited greatly from discussions with them. I am also thankful to my former advisor, Chih-Jen Lin, who equipped me with solid research skills.

In addition, I would like to acknowledge Ming-Wei Chang, Vivek Srikumar, and Rajhans Samdani for supporting me in the early stage of my Ph.D. studies. They always patiently answered my many questions. Mark Sammons taught me many things and has been always supportive. Eric Horn also assisted me in many aspects, especially in job finding. Nick Rizzolo, Alla Rozovskaya, Biplab Deka, Shyam Upadhyay, Lev Ratinov, He He, Akshay Krishnamurthy, Gourab Kundu, Haoruo Peng, Chen-Tse Tsai, and Ching-Pei Lee were collaborators in many of my projects, and I learned a lot from discussions with them. I would also like to thank my colleagues in the CogComp group, including Quang Do, Prateek Jindal, Jeff Pasternack, Dan Goldwasser, Parisa Kordjamshidi, Daniel Khasabi, Wei Lu, Stephen Mayhew, Subhro Roy, Yangqiu Song, Yuancheng Tu, Vinod Vydiswaran, John Weiting and Christos Christodoupoulos. Many thanks to all my friends from Taiwanese community in Champaign-Urbana for making my life colorful.

Finally, I would like to thank my parents and my girlfriend, Nanyun, for their sacrifice and support. Words cannot express how much I appreciate and love you.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ILP	Integer Linear Programming
NLP	Natural Language Processing
POS	Part-of-Speech Tagging
CRF	Conditional Random Field
SVM	Support Vector Machine
SSVM	Structured SVM
WSJ	Wall Street Journal
BMD	Block Minimization Algorithm
SBM	Selective Block Minimization
L^3M	Latent Left-Linking Model
PL^3M	Probabilistic Latent Left-Linking Model
DCD	Dual Coordinate Descent
DEMI-DCD	DEcoupled Model-update and Inference with Dual Coordinate Descent
AI-DCD	Amortized Inference framework for Dual Coordinate Descent
SGD	Stochastic Gradient Descent Algorithm

Chapter 1

Introduction

Machine learning techniques have been successfully applied in many fields, including natural language processing (NLP), computer vision, information retrieval, and data mining. Among these, classification methods and structured prediction models are two of the most widely used techniques. Standard classification methods are useful when a prediction problem can be formulated as one or several independent binary (True/False) or multiple-choice questions. Statistical learning techniques (e.g., support vector machine (Cortes and Vapnik, 1995) and logistic regression) have been widely studied for such problems. Many of these can be scaled up to handle relatively large amounts of data in applications like spam detection, object recognition, network intrusion detection, and human splice site recognition. On the other hand, many prediction problems consist of several inter-dependent binary or multiple-choice decisions. One can model the interrelated decision variables involved in these problems using *a structured object*, such as linear chain, clustering, or tree. Theses types of models are known as structured prediction models and are typically expressed as probabilistic models over a set of variables (e.g., Bayesian networks (Pearl, 1988), Markov random fields (Kindermann and Snell, 1980), factor graphs (Kschischang et al., 2001), and constrained conditional models (Chang et al., 2012b)). To estimate the parameters involved in these models from annotated data, many approaches have been proposed, including decomposed learning (Heckerman et al., 2000; Samdani and Roth, 2012), max-margin learning (Taskar et al., 2004; Joachims et al., 2009), and constraint-driven learning (Chang et al., 2012b). To focus our discussion, this thesis focuses on supervised discriminative learning methods for classification and structured prediction models.

During the last decade, the size of data sets used to build classifiers and structured predictors has steadily increased. The availability of large amounts of annotated data (e.g., for web applications such as spam filtering) provide great opportunities for research in machine learning and for

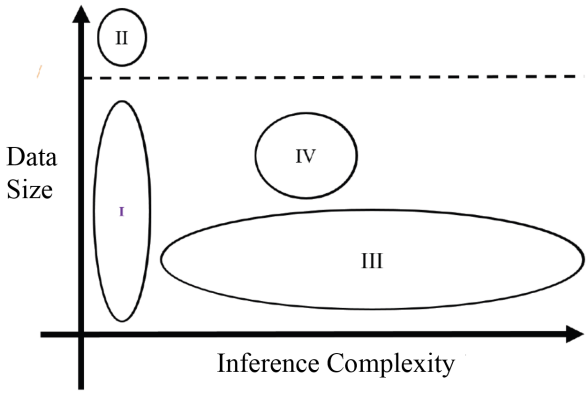


Figure 1.1: The size of the learning problems and the inference complexity of the model. This thesis focuses on the problems in regions II and III, though we will discuss an algorithm for learning latent representations of a knowledge base, where the learning problem is both large and complex (region IV). Prior work by the author of this thesis partly concerns problems in region I. The dashed line indicates a limitation on the size of the data that can be fully loaded into main memory, and where our region II is relative to it.

improving the quality of practical models, but also raise a scalability issue. Scalability is influenced by two key dimensions: the size of the data and the complexity of the decision structures. If the data is too large, training even a simple binary classifier can be difficult. For example, Yu et al. (2010) demonstrated a failure to train a binary classifier using Liblinear (Fan et al., 2008), a popular linear classification library, on a data set that was too large to be fully loaded into memory. When the data cannot fit into memory, batch learners, which load the entire data during the training process, suffer severely due to disk swapping (Yu et al., 2010). Even online learning methods, which deal with the memory problem by loading a portion of data at each time step, take considerable number of iterations to converge to a good model. Consequently, they require significant amount of disk access which slows down the training process. And, complex dependencies among the variables of interests are to be modeled, it is essential to consider an expressive model which makes exact learning and inference time consuming. In extreme cases, inferences over complex decision structures are practically intractable. Even where the decision structure is tractable, the time complexity of a typical inference algorithm is super-linear in the size of the output variables. As a result, the inference procedure often becomes a bottleneck in the learning process. Figure 1.1 depicts the scale of the machine learning problems and the focus of the thesis.

This thesis develops a conceptually unified approach, Selective Learning Algorithms, and, within it, presents several efficient learning algorithms that address the dimensions of scalability discussed above – large data sets and expressive dependencies among variables of interest. These approaches are based on an observation that a machine learning algorithm often consists of *many* small operations and sub-components. For example, when solving a binary classification problem with many training samples, one needs to update the model based on each sample. Each update takes a small amount of time, but in total, it contributes to the overall training time and, often, makes it much longer than possible or needed. Therefore, by identifying important components in the training process, we can devote more effort and resources (e.g., memory and computational power) to these *selected* components. We may even cache the computation on these components to save future computation, and, consequently, we may not need to attend to *all* components. This leads to a unified strategy that can be used to accelerate the training process.

In the rest of this chapter, we first present two motivating examples. Then, we provide more details about our approaches and contributions.

1.1 Motivating Examples and Challenges

In this section, we provide some examples that motivate the scalability issues in machine learning.

1.1.1 Large-Scale Classification

In the past few years, the size of data available for analysis has grown drastically. Many Internet companies report the need to learn on extremely large data sets, and consequently, benchmark data sets used by research increased significantly in size. For example, in one visual recognition challenge, ImageNet,[1] the learning task requires handling a data set of hundreds of gigabytes. Google reports that they have data sets with a hundred billion instances.[2] A learning task for recognizing a human acceptor splice (Sonnenburg and Franc, 2010) generates a dataset that requires 600 GB of memory to store. In some applications, unlimited amounts of annotated data can be generated. For example, we will show an example in Section 3.6 in which an infinite number of examples can be generated

[1] `http://www.image-net.org`

[2] `http://googleresearch.blogspot.com/2010/04/lessons-learned-developing-practical.html`

in order to learn a predictor for correct prepositions. Bottou[3] also shows that an infinite number of digit images can be generated from the MNIST dataset by pseudo-random deformations and translations (Loosli et al., 2007). All the above examples show the need for efficient and effective algorithms to handle large-scale data, and it has been argued that using more samples and adding more features (e.g, using n-gram in a bag-of-words model) actually boosts performance.

The challenge of training a model on large amounts of data is two-fold. First, as we mentioned above, although the computation involved in updating the model on each example is small, this accumulates, when dealing with large amounts of data, and becomes very significant. Second, when the data is large, memory access often becomes the bottleneck. The situation is even worse when the data is too large to fit in memory. In Chapter 3, we will show that by selecting informative instances during the training, we can significantly reduce the training cost on large-scale data.

1.1.2 Supervised Clustering: Coreference Resolution

Many machine learning applications, including spam filtering (Haider et al., 2007), network intrusion detection (Guha et al., 2003), and thread detection in text messages (Shen et al., 2006), need to cluster items based on their similarity. Different from transitional unsupervised clustering problems, where evaluating clustering is often a key issue, the clusters in these clustering problems have clear semantics. Therefore it is possible to obtain supervised data, which can be used to train and evaluate the clustering models.

In the following, we use an NLP task as a motivating example to demonstrate the scalability problems. Consider the following paragraph and the question:

> (ENGLAND, June, 1989) - [**Christopher Robin**] is alive and well. He lives in England. He is the same person that you read about in the book, Winnie the Pooh. As a boy, Chris lived in a pretty home called Cotchfield Farm. When [**Chris**] was three years old, his father wrote a poem about him. The poem was printed in a magazine for others to read. [**Mr. Robin**] then wrote a book. He made up a fairy tale land where Chris lived. His friends were animals. There was a bear called Winnie the Pooh. There was also an owl and a young pig, called a piglet. All the animals were stuffed toys that

[3] `http://leon.bottou.org/projects/infimnist`

Chris owned.

Q: Do **[Christopher Robin]**, **[Chris]** and **[Mr. Robin]** (the boldfaced mentions) refer to the same person?

At the first glance, one might think that **[Chris]** and **[Mr. Robin]** both refer to **[Christopher Robin]**. However, after carefully reading the paragraph, we know that **[Mr. Robin]** refers to **his father** in the fourth line, and **his** refers to **[Chris]**. Therefore, **[Mr. Robin]** refers to Chris's father and cannot be the same as **[Chris]**. This suggests that in order to correctly answer this question, we need to consider the complex decision structure, and simultaneously take several pairwise local decisions (e.g., does **[Mr. Robin]** refer to **his father**?) into account.

The general setting of this problem is called *coreference resolution*, where we want to group the noun phrases (mentions) into equivalence classes, so that the mentions in the same class refer to the same entity in the real world. For example, all the underlined mentions in the above paragraph refer to the same entity, **Christopher Robin**. Therefore, they belong to the same group. We consider a supervised learning setting, where we assume a collection of articles with coreference annotations. The goal is to learn a model that predicts the coreference clusters for a previously unseen document. A natural way to model this problem is to break the coreference clustering decision into pairwise local decisions, which estimate whether two mentions can be put into the same coreference cluster. As shown in the example above, these local predictions are mutually dependent and have to obey the property of transitivity. Therefore, the inference procedure requires global consideration of all assignments simultaneously, resulting in a computational difficulty. This is a typical scalability issue when dealing with problems having a complex structure. We will provide more details in Chapters 2 and 6.

1.2 Selective Algorithms for Learning at Scale

Various approaches have been proposed to train models on large and complex data. Most approaches have focused on designing optimization methods (Joachims, 2006; Shalev-Shwartz et al., 2007; Taskar et al., 2005; Chang and Yih, 2013; Lacoste-Julien et al., 2013) and parallel algorithms (Woodsend and Gondzio, 2009; Chang et al., 2007a; Agarwal et al., 2014; McDonald et al., 2010a). In addition, online learning algorithms, which update based on only one instance at a time,

have been shown to be capable of handling large-scale data (Crammer et al., 2006; Dredze et al., 2008; Crammer et al., 2009; Collins, 2002) as well.

In this thesis, we will explore another orthogonal direction – saving computation by selecting important components, such as instances, candidate structures, and latent items, and devoting more effort to them during the training. The informative components can be predefined but are often selected by the model. We will demonstrate that this idea can be applied when designing efficient learning algorithms as well as when modeling problems involving complex decisions. In all cases, the training time is reduced without compromising the performance of the model. In some cases, the performance even improves (see Chapter 6). We will provide sound theoretical justifications of the selection framework. Specifically, from an optimization perspective, selecting informative samples and candidate structures can be treated as solving the learning problem using block coordinate descent in the dual form. From a modeling perspective, selecting latent items is related to the latent structured models (Quattoni et al., 2007; Yu and Joachims, 2009; Samdani et al., 2012b).

Some earlier approaches in the literature have similarities to some aspects of the approach presented in this thesis. Selecting informative components during the training has been discussed in the literature, such as working set selection and the shrinking strategy for kernel SVM (Keerthi et al., 2001; Fan et al., 2005; Joachims, 1999), caching for Perceptron (Collins and Roark, 2004), emphasizing scheme (Guyon et al., 1992), and selective sampling for training on large data (Pérez-Cruz et al., 2004; Tong and Koller, 2002; Yu et al., 2003). Dual coordinate descent and sequential minimal optimization have been widely discussed especially for solving the SVM problem and its variants (e.g., (Platt, 1999; Hsieh et al., 2008; Shalev-Shwartz and Zhang, 2013)). We will provide more discussions in the related work section in each chapter.

1.3 Overview of This Work

This thesis addresses scalability problems, such as those discussed in previous sections. We derive efficient selective algorithms for three different scenarios:

- selecting informative samples in large-scale classification (Chapter 3);

- designing a parallel algorithm to select candidate structures while learning a structured model (Chapter 4) and
- selecting and caching inference solutions in learning a structured model (Chapter 5); and
- selecting latent items for online clustering models (Chapter 6).

We provided a summary of our contributions below.

Learning with Large-Scale Data. In Chapter 3, we propose a selective block minimization (SBM) framework (Chang and Roth, 2011) for data that cannot fit in memory. The solver loads and trains on one block of samples at a time and caches informative samples in memory during the training process. Thus, it requires fewer data loads before reaching a given level of performance relative to existing approaches. By placing more emphasis on learning from the data that is already in memory, the method enjoys provably fast global convergence and reduces training time, in some problems, from half a day to half an hour. It can even obtain a model that is as accurate as the optimal one with just a single pass over the data set. For the same data, the best previously published method (Yu et al., 2010) requires loading the data from disk 10 times to reach the same performance.

Learning and Inference for General Structured Prediction Models. In many cases, decision making involves predicting a set of interdependent outputs. Structured prediction models have been proposed to model these interdependencies, and have been shown to be powerful. However, their high computational cost often causes a scalability problem. In this thesis, we explore methods which select and cache candidate structures when learning structured prediction models. We show how to improve structured training algorithms from two orthogonal directions – parallelism and amortization.

In Chapter 4, we introduce the DEMI-DCD algorithm (DEcoupled Model-update and Inference with Dual Coordinate Descent) (Chang et al., 2013b), a new barrier-free parallel algorithm based on the dual coordinate descent (DCD) method for the L2-loss structural SVM, which is a popular structured prediction model. DEMI-DCD removes the need for a barrier between a model update phase and an inference phase involved in the training, allowing us to distribute these two phases across multiple cores. DEMI-DCD is three times faster than traditional master-slave parallel learning

methods, while maintaining sound convergence properties.

In Chapter 5, we observed that training a structured prediction model involves performing several inference steps. Over the lifetime of training, many of these inference problems, although different, share the same solution. We demonstrate that by exploiting this redundancy of solutions, a solver can reduce by 76% – 90% the number of inference calls needed without compromising the performance of the model, thus significantly reducing the overall training time (Chang et al., 2015).

Learning and Inference with a Highly Complex Structure. Many applications involve a highly complex structure. In such situations, learning and inference are intractable. In Chapter 6, we show that the idea of selection can be used to model such a problem. Specifically, we design an auxiliary structure that simplifies a complex structured prediction model for on-line supervised clustering problems. For each item, the structure selects only one item (in this case, the one to its left in the mention sequence) and uses it to update the pairwise metric (a feature-based pairwise item similarity function), assuming a latent left linking model (L^3M). We derive efficient learning and inference algorithms for L^3M and apply them to the coreference resolution problem, which is aimed at clustering denotative phrases ("mentions") that refer to the same entity into an equivalent class. This leads to an efficient and principled approach to the problem of on-line clustering of streaming data. In particular, when applied to the problem of coreference resolution, it provides a linguistically motivated approach to this important NLP problem (Chang et al., 2013a). This was the best performing coreference system at the time of publication, and the approach has become mainstream and inspired follow-up work (Ma et al., 2014; Lassalle and Denis, 2015).

Chapter 2

Background

Statistical machine learning approaches have been widely used in many applications, including predicting linguistic structure and semantic annotation. In this chapter, we briefly introduce the corresponding notation and the necessary background knowledge. We mainly focus on the general concepts used in classification and structured prediction, where training a model is time-consuming.

Table 2.1 lists the notation used throughout this thesis. The goal of a machine learning algorithm is to learn a mapping function $f(\boldsymbol{x}; \boldsymbol{w})$ from an input space $\mathcal{X}$ to an output space $\mathcal{Y}$. Depend on how a learning problem is modeled, the output variable of interest can be binary (i.e., $y \in \{1, -1\}$), categorical ($y \in \{1, 2, 3, \ldots, k\}$), or structured ($\boldsymbol{y} = (y_1, y_2, \ldots, y_j, \ldots y_k)$, where $y_j \in \mathcal{Y}_j$, a discrete set of candidates for the output variable y_j, and $\boldsymbol{y} \in \mathcal{Y}$, a set of feasible structures). This mapping function can be parameterized with weights $\boldsymbol{w}$ of task-relevant features $\phi(\cdot)$. If the output variable is binary or categorical, the features are typically defined only on input, and can be represented as a feature vector $\phi(\boldsymbol{x})$. For notational simplicity in this thesis, we use $\boldsymbol{x}$ instead of $\phi(\boldsymbol{x})$ to represent this feature vector. If the output variable is modeled as a structured object, the feature vector $\phi(\boldsymbol{x}, \boldsymbol{y})$ is often defined on both input and output, allowing us to take into account the interdependency between variables in the structured object.

We assume there is an underlining distribution $P(\boldsymbol{x}, \boldsymbol{y})$ to the data. Unless stated otherwise, in a supervised learning setting, we assume that a set of independent and identically distributed (i.i.d.) training data $\mathcal{D} = (\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=1}^{l}$ is given. The goal of supervised learning is to estimate the mapping function using the training data, such that the expected loss is minimal:

$$E[f] = \int_{\mathcal{X} \times \mathcal{Y}} \Delta(f(\boldsymbol{x}; \boldsymbol{w}), \boldsymbol{y}) dP(\boldsymbol{x}, \boldsymbol{y}), \tag{2.1}$$

where $\Delta(f(\boldsymbol{x}; \boldsymbol{w}), \boldsymbol{y})$ is a loss function estimating the difference between two output objects. If the

Notation	Description
$\mathcal{D}$	Dataset
$\boldsymbol{x} \in \mathbb{R}^n$	Input instance
$y \in \{1, -1\}$	Output (binary)
$y \in \{1, 2, 3, \ldots, k\}$	Output (multiclass)
$\boldsymbol{y} \in \mathcal{Y}$	Structured Output
$\phi(\cdot)$	Feature generating function
$\boldsymbol{w} \in \mathbb{R}^d$	Weight vector
$\Delta(\boldsymbol{y}', \boldsymbol{y})$	Distance between structures $\boldsymbol{y}'$

Table 2.1: Notation.

output variable is binary or categorical, Δ is usually defined as a zero-one loss,

$$\Delta(y, y') = I(y \neq y'), \tag{2.2}$$

where $I(\cdot)$ is an indicator function. If the output is structured, a common choice of the loss function is Hamming loss $\Delta(\boldsymbol{y}, \boldsymbol{y}') = \sum_i I(y_i \neq y'_i)$, where y_i is the sub-component of $\boldsymbol{y}$.

Once the mapping function is learned, we proceed to the test phase. Given a test set $D_{\text{test}} = (\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=l+1}^{l+m}$ drawn i.i.d from the same distribution, we apply the learned function to predict the output value of $\boldsymbol{x}_i \in D_{\text{test}}$, comparing the prediction with $\boldsymbol{y}_i$ to evaluate performance. In the following, we provide details about how this mapping function f is defined, how it is used in prediction, and how the weights are learned.

2.1 Binary Classification

We begin by discussing binary classification models. We mainly focus on linear classification models, which have been shown to handle large amounts of data well.

In a binary classification problem, the output is a binary variable $y \in \{1, -1\}$. Given a set of data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{l}, \boldsymbol{x}_i \in R^n, y_i = \{1, -1\}$, the goal of linear classification models is to learn a linear function,

$$f(\boldsymbol{x}; \boldsymbol{w}) = sign(\boldsymbol{w}^T \boldsymbol{x}), \tag{2.3}$$

to predict the label y of an instance $\boldsymbol{x}$. Several learning algorithms have been proposed and studied for estimating the weights $\boldsymbol{w}$ in (2.3) (e.g., (Minsky and Papert, 1969; Platt, 1999; Ng and Jordan,

2001)). In the following, we discuss a popular approach, linear Support Vector Machines (linear SVM) (Cortes and Vapnik, 1995), which has been shown to perform well in dealing with large data (Shalev-Shwartz et al., 2007; Hsieh et al., 2008; Joachims, 2006).

2.1.1 Linear Support Vector Machines

Given a set of training data $D = \{\boldsymbol{x}_i, y_i\}_{i=1}^{l}$, Linear SVM considers the following minimization problem:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l}\max(1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i, 0), \tag{2.4}$$

where the first term is a regularization term, the second term is a loss term, and $C > 0$ is a penalty parameter that balances these two terms. The loss function in (2.4) is called a surrogate hinge loss function, which is a convex upper bound of the zero-one loss (i.e., Eq. (2.2)). Other surrogate loss functions that can be used for binary classification include square loss, square-hinge loss, and logistic loss.

Instead of solving (2.4), one may solve the equivalent dual problem:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & f(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha} \\ \text{s.t.} \quad & 0 \le \alpha_i \le C, i = 1, \ldots, l, \end{aligned} \tag{2.5}$$

where $\boldsymbol{e} = [1, \ldots, 1]^T$ and $Q_{ij} = y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$. Eq. (2.5) is a box constraint optimization problem, and each variable corresponds to one training instance. Let $\boldsymbol{\alpha}^*$ be the optimal solution of (2.5). We call an instance $(\boldsymbol{x}_i, y_i)$ a support vector if the corresponding parameter satisfies $\alpha_i^* > 0$, and an instance i is an unbounded support vector if $0 < \alpha_i^* < C$. We show later in Chapter 3 that unbounded support vectors are more important than the other instances during the training process.

In the dual coordinate descent method (Hsieh et al., 2008), we cyclically update each element in $\boldsymbol{\alpha}$. To update α_i, we solve the following one-variable sub-problem:

$$\begin{aligned} \min_{d} \quad & f(\boldsymbol{\alpha} + d\boldsymbol{e}_i) \\ \text{s.t.} \quad & 0 \le \alpha_i + d \le U, i = 1, \ldots, l. \end{aligned} \tag{2.6}$$

$\boldsymbol{e}_i$ is a vector, where the i-th element is 1 and the rest are 0. The objective function $f(\boldsymbol{\alpha} + d\boldsymbol{e}_i)$

Algorithm 1 A dual coordinate descent method for Linear SVM

Require: Dataset $\mathcal{D}$
Ensure: The learned model $\boldsymbol{w}$.
1: Initialize $\boldsymbol{\alpha} = \mathbf{0}, \boldsymbol{w} = \mathbf{0}$.
2: **for** $t = 1, 2, \ldots$ (outer iteration) **do**
3: Randomly shuffle the data.
4: **for** $j = 1, \ldots, m$ (inner iteration) **do**
5: $G = y_i \boldsymbol{w}^T \boldsymbol{x}_i - 1$
6: $\bar{\alpha}_i \leftarrow \alpha_i$
7: $\alpha_i \leftarrow \min(\max(\alpha_i - G/(\boldsymbol{x}_i^T \boldsymbol{x}_i), 0), C)$
8: $\boldsymbol{w} \leftarrow \boldsymbol{w} + (\alpha_i - \bar{\alpha}_i) y_i \boldsymbol{x}_i$
9: **end for**
10: **end for**

of Eq. (2.6) is equivalent to $\frac{1}{2} Q_{ii} d^2 + \nabla_i f(\boldsymbol{\alpha}) d + \text{constant}$, and $\nabla_i f(\boldsymbol{\alpha}) = (Q\boldsymbol{\alpha})_i - 1$ is the i-th element of the gradient of f at $\boldsymbol{\alpha}$. Solving Eq. (2.6), we obtain the following update rule:

$$\alpha_i = \min\left(\max\left(\alpha_i - \frac{\nabla_i f(\boldsymbol{\alpha})}{Q_{ii}}, 0\right), C\right). \tag{2.7}$$

Note that by defining

$$\boldsymbol{w} = \sum_{j=1}^{l} y_j \alpha_j \boldsymbol{x}_j, \tag{2.8}$$

$\nabla_i f(\boldsymbol{\alpha})$ can be rewritten as $y_i \boldsymbol{w}^T \boldsymbol{x}_i - 1$. Therefore, the complexity of Eq. (2.7) is linear to the number of average active features. Algorithm 1 summarizes the algorithm.

Hsieh et al. (2008) show that Algorithm 1 is very efficient. When data can be fully loaded into memory, training a linear model on sparse data with millions of samples and features takes only a few minutes. However, when the data set is too large, learning a model is difficult. We will discuss this scenario in Chapter 3.

2.2 Structured Prediction

Many prediction problems in natural language processing, computer vision, and other fields are structured prediction problems, where decision making involves assigning values to interdependent variables. The output structure can represent sequences, clusters, trees, or arbitrary graphs over the decision variables.

The goal of structured prediction is to learn a scoring function,

$$S(\boldsymbol{y}; \boldsymbol{w}, \boldsymbol{x}) = \boldsymbol{w}^T \phi(\boldsymbol{x}, \boldsymbol{y}),$$

that assigns a higher score to a better structure. As we described above, the scoring function can be parameterized by a weight vector $\boldsymbol{w}$ of features $\phi(\boldsymbol{x}, \boldsymbol{y})$ extracted from both input $\boldsymbol{x}$ and output structure $\boldsymbol{y}$. This definition of a feature function allows us to design features to express the dependency between the local output variables for instances. In general, designing feature templates requires domain knowledge. For example, in natural language processing, bag-of-words features (words in some window surrounding the target word) are commonly used.

For example, the coreference resolution problem introduced in Chapter 1 is a structured prediction problem, where an input is an article and the output is a clustering of mentions representing the coreference chains. Formally, we can represent the clustering of m mentions, using a set of binary variables $\boldsymbol{y} = \{y_{u,v} \mid 1 \leq u, v \leq m\}$, where $y_{u,v} = 1$ if and only if mentions u and v are in the same cluster. The assignments of these binary variables have to obey the transitivity property (i.e., for any u, v, t, if $y_{u,v} = 1$ and $y_{v,t} = 1$, then $y_{u,t} = 1$); therefore, these assignments are mutually dependent. In order to define the global decision score, we can design a feature-based pairwise classifier to assign a compatibility score $c_{u,v}$ to a pair of mentions (u, v). Then, we can take these pairwise mention scores over a document as input, aggregating them into clusters representing clustering scores. If we consider scoring a clustering of mentions by including all possible pairwise links in the score, the global scoring function can be written as:

$$S(\boldsymbol{y}; \boldsymbol{w}, \boldsymbol{x}) = \sum_{u,v} y_{u,v} c_{u,v} = \sum_{u,v} y_{u,v} \boldsymbol{w}^T \phi(u, v), \tag{2.9}$$

where $\phi(u, v)$ represents features defined on pairs of mentions.[1] In such a way, only the scores associated with mention pairs within the cluster contribute to the global clustering score. We refer to this approach as the All-Link coreference model.

In the remainder of this section, we will first describe how to make predictions using a global scoring function (inference). Then, we will discuss how to estimate the parameters $\boldsymbol{w}$ based on

[1] For simplicity of modeling, our features do not depend on the output clustering.

the training data (learning). We continue using coreference resolution as a running example in the following description.

2.2.1 Inference for Structured Models

The inference problem in a structured prediction framework is to find the *best structure* $\boldsymbol{y}^* \in \mathcal{Y}$ for $\boldsymbol{x}$ according to a given model $\boldsymbol{w}$,

$$\boldsymbol{y}^* = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \quad \boldsymbol{w}^T \phi(\boldsymbol{x}, \boldsymbol{y}), \tag{2.10}$$

where $\mathcal{Y}$ represents a set of feasible output structures. The output space $\mathcal{Y}$ is usually determined by the structure of an output object. For example, in a coreference resolution problem, the transitivity property of clustering structures restricts the possible local assignments. Domain specific constraints could be added to further restrict the possible output space (Roth and Yih, 2007). Injecting domain knowledge using declarative constraints has been widely used in natural language processing(Roth and Yih, 2007; Clarke and Lapata, 2008; Koo et al., 2010). We will show an example in the context of coreference later in Chapter 6. Overall, Eq. (2.10) is a discrete constrained optimization problem. In the following, we briefly review the techniques for solving Eq. (2.10).

Roth and Yih (2004) proposed to formalize Eq. (2.10) as an integer linear programming (ILP) problem. The use of the ILP formulation has been popular in natural language processing, vision, and other areas (Roth and Yih, 2004; Clarke and Lapata, 2006; Riedel and Clarke, 2006). Taking the All-Link approach we mentioned above as an example, All-Link inference finds a clustering by solving the following ILP:

$$\begin{aligned} \arg\max_{\mathbf{y}} \quad & \sum_{u,v} c_{uv} y_{uv} \\ \text{s.t.} \quad & y_{uq} \geq y_{uv} + y_{vq} - 1 \ \forall u, v, q \\ & y_{uv} \in \{0, 1\} \ \forall u, v. \end{aligned} \tag{2.11}$$

The objective function of Eq. (2.11) is the sum of all pairs within the same cluster (Eq. (2.9)). The inequality constraints enforce the transitive closure of the clustering. The solution of Eq. (2.11) is a set of cliques, and mentions in the same cliques corefer.

If the local decisions involved in the structured prediction are categorical, one can use a binary vector $\boldsymbol{z} = \{z_{j,a} \mid j = 1 \cdots N, a \in \mathcal{Y}_j\}, \boldsymbol{z} \in \mathcal{Z} \subset \{0,1\}^{\sum |\mathcal{Y}_i|}$ to represent the output variable $\boldsymbol{y}$, where $z_{j,a}$ denotes whether or not y_j assumes the value a or not. This way, (2.10) can be reformulated as a binary ILP:

$$\boldsymbol{z}^* = \arg\max_{\boldsymbol{z} \in \mathcal{Z}} \quad \boldsymbol{c}^T \boldsymbol{z}, \tag{2.12}$$

where $\boldsymbol{c}^T \boldsymbol{z} = \boldsymbol{w}^T \phi(\boldsymbol{x}_i, \boldsymbol{y})$, $\sum_a z_{j,a} = 1$, and $c_{j,a}$ is the coefficient corresponding to $z_{j,a}$.

Although there is no existing polynomial-time algorithm for solving ILP problems, there are some off-the-shelf ILP solvers (e.g., Gurobi(Gurobi Optimization, Inc., 2012)) available, which can handle problems with small scale in a reasonable time (e.g., in a few seconds).

As a remark, some specific output structures (e.g., linear chain structures) allow the inference to be cast as a search problem, for which efficient exact algorithms are readily available (e.g., the Viterbi algorithm). In addition, many approximate inference algorithms, such as LP relaxation (Schrijver, 1986; Sontag, 2010), Lagrangian relaxation, dual decomposition (Held and Karp, 1971; Rush et al., 2010; Rush and Collins, 2002), message passing, belief propagation (Pearl, 1988; Yedidia et al., 2005)), have been proposed to address the intractability of inference.

2.2.2 Learning Structured Prediction Models

In this section, we discuss various approaches for learning structured prediction models. The goal is to find the best model $\boldsymbol{w}$ that minimizes the expected loss (2.1).

Decoupling Learning from Inference. The simplest way to learn a structured prediction model is to decouple the learning from inference. In the training phase, all local classifiers are learned independently. The structured consistency is applied only in the test phase. Punyakanok et al. (2005b) call this approach *learning plus inference* (L+I). This approach is in contrast with *inference based learning* (IBT), where local classifiers are learned jointly by using global feedback provided from the inference process. Although L+I is simple and has shown success in many applications (Punyakanok et al., 2005a), Punyakanok et al. (2005b) show that when the local classifier is hard to learn and the amount of structured data is sufficiently large, IBT outperforms L+I. We will show an example in Chapter 6 in which joint learning is important for building a corefernece resolution system. In the following, we discuss several IBT approaches.

Structured SVM. Training a structured SVM (SSVM) (Tsochantaridis et al., 2005) is framed as the problem of learning a real-valued weight vector $\boldsymbol{w}$ by solving the following optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w},\boldsymbol{\xi}} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_i \ell(\xi_i) \\ s.t. \quad & \boldsymbol{w}^T\phi(\boldsymbol{x}_i,\boldsymbol{y}_i) - \boldsymbol{w}^T\phi(\boldsymbol{x}_i,\boldsymbol{y}) \geq \Delta(\boldsymbol{y}_i,\boldsymbol{y}) - \xi_i, \quad \forall i, \boldsymbol{y} \in \mathcal{Y}_i, \end{aligned} \tag{2.13}$$

where $\phi(\boldsymbol{x},\boldsymbol{y})$ is the feature vector extracted from input $\boldsymbol{x}$ and output $\boldsymbol{y}$, and $\ell(\xi)$ is the loss that needs to be minimized. The constraints in (2.13) indicate that for all training examples and all possible output structures, the score for the correct output structure $\boldsymbol{y}_i$ is greater than the score for other output structures $\boldsymbol{y}$ by at least $\Delta(\boldsymbol{y}_i,\boldsymbol{y})$. The slack variable $\xi_i \geq 0$ penalizes the violation. The loss ℓ is an increasing function of the slack that is minimized as part of the objective: when $\ell(\xi) = \xi$, we refer to (2.13) as an L1-loss structured SVM, while when $\ell(\xi) = \xi^2$, we call it an L2-loss structured SVM[2]. For mathematical simplicity, in this thesis, we only consider the linear L2-loss structured SVM model, although our method can potentially be extended to other variants of the structured SVM.

Instead of directly solving (2.13), several optimization algorithms for SSVM consider its dual form (Tsochantaridis et al., 2005; Joachims et al., 2009; Chang and Yih, 2013) by introducing dual variables $\alpha_{i,\boldsymbol{y}}$ for each output structure $\boldsymbol{y}$ and each example $\boldsymbol{x}_i$. If $\boldsymbol{\alpha}$ is the set of all dual variables, the dual problem can be stated as

$$\min_{\boldsymbol{\alpha}\geq 0} \quad D(\boldsymbol{\alpha}) \equiv \frac{1}{2}\left\|\sum_{\alpha_{i,\boldsymbol{y}}} \alpha_{i,\boldsymbol{y}}\phi(\boldsymbol{y},\boldsymbol{y}_i,\boldsymbol{x}_i)\right\|^2 + \frac{1}{4C}\sum_i\left(\sum_{\boldsymbol{y}}\alpha_{i,\boldsymbol{y}}\right)^2 - \sum_{i,\boldsymbol{y}}\Delta(\boldsymbol{y},\boldsymbol{y}_i)\alpha_{i,\boldsymbol{y}}, \tag{2.14}$$

where $\phi(\boldsymbol{y},\boldsymbol{y}_i,\boldsymbol{x}_i) = \phi(\boldsymbol{y}_i,\boldsymbol{x}_i) - \phi(\boldsymbol{y},\boldsymbol{x}_i)$. The constraint $\boldsymbol{\alpha} \geq 0$ restricts all the dual variables to be non-negative (i.e., $\alpha_{i,\boldsymbol{y}} \geq 0, \ \forall i,\boldsymbol{y}$).

For the optimal values, the relationship between the primal optimal $\boldsymbol{w}^*$ (that is, the solution

[2]In L2-loss structured SVM formulation, one may replace $\Delta(\boldsymbol{y}_i,\boldsymbol{y})$ by $\sqrt{\Delta(\boldsymbol{y}_i,\boldsymbol{y})}$ to obtain an upper bound on the empirical risk (Tsochantaridis et al., 2005). However, we keep using $\Delta(\boldsymbol{y}_i,\boldsymbol{y})$ for computational and notational convenience. Thus, Eq. (2.13) minimizes the mean square loss with a regularization term.

of (2.13)), and the dual optimal $\boldsymbol{\alpha}^*$ (that is, the solution of (2.14)) is

$$\boldsymbol{w}^* = \sum_{i,\boldsymbol{y}} \alpha^*_{i,\boldsymbol{y}} \phi(\boldsymbol{y}, \boldsymbol{y}_i, \boldsymbol{x}_i).$$

Although this relationship only holds for the solutions, in a linear model, one can maintain a temporary vector,

$$\boldsymbol{w} \equiv \sum_{i,\boldsymbol{y}} \alpha_{i,\boldsymbol{y}} \phi(\boldsymbol{y}, \boldsymbol{y}_i, \boldsymbol{x}_i), \tag{2.15}$$

to assist the computations (Hsieh et al., 2008).

In practice, for most definitions of structures, the set of feasible structures for a given instance (that is, $\mathcal{Y}_i$) is exponentially large, leading to an exponentially large number of dual variables. Therefore, existing dual methods (Chang and Yih, 2013; Tsochantaridis et al., 2005) maintain an active set of dual variables $\mathcal{A}$ (also called the working set in the literature). During training, only the dual variables in $\mathcal{A}$ are considered for an update, while the rest $\alpha_{i,\boldsymbol{y}} \notin \mathcal{A}$ are fixed to 0. We denote the active set associated with the instance $\boldsymbol{x}_i$ as $\mathcal{A}_i$ and $\mathcal{A} = \bigcup_i \mathcal{A}_i$.

To select a dual variable for the update, DCD solves the following optimization problem for each example $\boldsymbol{x}_i$:

$$\begin{aligned} \bar{\boldsymbol{y}} &= \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \quad (-\nabla D(\boldsymbol{\alpha}))_{i,\boldsymbol{y}} \\ &= \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \quad \boldsymbol{w}^T \phi(\boldsymbol{x}_i, \boldsymbol{y}) + \Delta(\boldsymbol{y}_i, \boldsymbol{y}) \end{aligned} \tag{2.16}$$

and adds $\alpha_{i,\bar{\boldsymbol{y}}}$ into $\mathcal{A}$ if

$$-\nabla D(\boldsymbol{\alpha})_{i,\bar{\boldsymbol{y}}} > \delta, \tag{2.17}$$

where δ is a user-specified tolerance parameter. Eq. (2.16) is often referred to as a loss-augmented inference problem. In practice, one may choose a loss $\Delta(\boldsymbol{y}_i, \boldsymbol{y})$ such that Eq. (2.16) can be formulated as an ILP problem (2.12).

Structured Perceptron. The Structured Perceptron (Collins, 2002) algorithm has been widely used. At each iteration, it picks an example $\boldsymbol{x}_i$ that is annotated with $\boldsymbol{y}_i$ and finds its best structured output $\bar{\boldsymbol{y}}$ according to the current model $\boldsymbol{w}$, using an inference algorithm. Then, the

model is updated as

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(\phi(\boldsymbol{x}_i, \boldsymbol{y}_i) - \phi(\boldsymbol{x}_i, \bar{\boldsymbol{y}})),$$

where η is a learning rate. Notice that the Structured Perceptron requires an inference step before each update. This makes it different from the dual methods for structured SVMs where the inference step is used to update the active set. The dual methods have the important advantage that they can perform multiple updates on the elements in the active set without doing inference for each update. As we will show in Section 4.3, this limits the efficiency of the Perceptron algorithm. Some caching strategies have been developed for structured Perceptron. For example, Collins and Roark (2004) introduced a caching technique to periodically update the model with examples on which the learner had made mistakes in previous steps. However, this approach is ad-hoc, without convergence guarantees.

Conditional Random Field. Conditional Random Field (CRF) (Lafferty et al., 2001) is a discriminative probabilistic model that models the condition probability of output $\boldsymbol{y}$ given an input $\boldsymbol{x}$ as

$$Pr(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{w}) = \frac{\exp(\boldsymbol{w}^T \phi(\boldsymbol{x}, \boldsymbol{y}))}{\sum_{\boldsymbol{y} \in \mathcal{Y}} \exp(\boldsymbol{w}^T \phi(\boldsymbol{x}, \boldsymbol{y}))}. \tag{2.18}$$

The model $\boldsymbol{w}$ can be estimated by minimizing the regularized negative conditional log-likelihood,

$$\min_{\boldsymbol{w}} \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w} + C \sum_{i=1}^{l} \left(\log \left(\sum_{\boldsymbol{y} \in \mathcal{Y}} \exp(\boldsymbol{w}^T \phi(\boldsymbol{x}_i, \boldsymbol{y})) \right) - \boldsymbol{w}^T \phi(\boldsymbol{x}_i, \boldsymbol{y}_i) \right). \tag{2.19}$$

If we rewrite Eq. (2.13) with L1-Loss:

$$\min_{\boldsymbol{w}, \boldsymbol{\xi}} \quad \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w} + C \sum_{i=1}^{l} \left(\max_{\boldsymbol{y} \in \mathcal{Y}} \left(\boldsymbol{w}^T \phi(\boldsymbol{x}_i, \boldsymbol{y}_i) + \Delta(\boldsymbol{y}_i, \boldsymbol{y}) \right) - \boldsymbol{w}^T \phi(\boldsymbol{x}_i, \boldsymbol{y}) \right). \tag{2.20}$$

Then, we can see that the key difference between CRF and Structured SVM is the use of the loss function.

The denominator of Eq. (2.18), $\sum_{\boldsymbol{y} \in \mathcal{Y}} \exp(\boldsymbol{w}^T \phi(\boldsymbol{x}, \boldsymbol{y}))$, is called a partition function. To solve Eq. (2.19), we have to efficiently estimate the value of the partition function. For some specific structured prediction problems, this function can be estimated by a dynamic programming algorithm (e.g., the forward algorithm for a linear chain structure). However, in general, estimating

the partition function is hard. We will discuss a probabilistic model related to CRF in Chapter 6.

Chapter 3

Training Large-scale Linear Models With Limited Memory

In the past few years, many companies reported the need to learn on extremely large data sets. Although most of data sets can be stored in a single machine, some of them are too large to be stored in the memory. When data cannot be fully loaded in memory, the training time consists of two parts: (1) the time used to update model parameters given the data in memory and (2) the time used to load data from disk. The machine learning literature usually focuses on the first – algorithms' design – and neglects the second. Consequently, most algorithms aim at reducing the CPU time and the number of passes over the data needed to obtain an accurate model (e.g., (Shalev-Shwartz et al., 2007; Hsieh et al., 2008)). However, the cost of data access from disk could be an order of magnitude higher. For example, a popular scheme that addresses the memory problem is to use an online learner to go over the entire data set in many rounds (e.g., (Langford et al., 2009a)). However, an online learner, which performs a simple weight update on one training example at a time, usually requires a large number of iterations to converge. Therefore, when an online learner loads data from the disk at each step, disk I/O bound becomes the bottleneck of the training process. Moreover, online methods consume a little memory as they consider only one sample at a time. Because a modern computer has a few gigabytes of memory, the amount of disk access can be reduced if the memory capacity is fully utilized.

In this chapter, *we focus on designing simple but efficient algorithms that place more effort on learning from the data that is already in memory. This will significantly speed up the training process on data sets larger than what can be held in memory.*

3.1 A Selective Block Minimization Algorithm

In (Chang and Roth, 2011), we developed a selective block minimization (SBM) algorithm which selects and caches informative samples during learning. We describe the SBM method by exemplifying it in the context of an L1-loss linear SVM (Eq. (2.4)) for binary classification.

With abundant memory, one can load the entire data set and solve (2.5) with a batch learner (Hsieh et al., 2008) as we discussed in Section 2.1.1. However, when memory is limited, only part of the data can be considered at a given time. Yu et al. (2010) apply a block minimization technique to deal with this situation. The solver splits the data into blocks $B_1, B_2, \dots B_m$ and stores them in compressed files. Initializing $\boldsymbol{\alpha}$ to zero vectors, it generates a sequence of solutions $\{\boldsymbol{\alpha}^{t,j}\}_{t=1\dots\infty, j=1\dots m}$. We refer to the step from $\boldsymbol{\alpha}^t = \boldsymbol{\alpha}^{t,1}$ to $\boldsymbol{\alpha}^{t+1} = \boldsymbol{\alpha}^{t,m+1}$ as an *outer* iteration, and $\boldsymbol{\alpha}^{t,j}$ to $\boldsymbol{\alpha}^{t,j+1}$ as an *inner* iteration. For updating from $\boldsymbol{\alpha}^{t,j}$ to $\boldsymbol{\alpha}^{t,j+1}$, the block minimization method soling a sub-problem on B_j. By loading and training on a block of samples at a time, it employs a better weight update scheme and achieves faster convergence than an online algorithm. However, since informative samples are likely to be present in different blocks, the solver wastes time on unimportant samples.

Intuitively, we should select only informative samples and use them to update the model. However, trying to select only informative samples and using only them in training may be too sensitive to noise and initialization and does not yield the appropriate convergence result we show later.

Algorithm 2 A Selective Block Minimization Algorithm for Linear Classification

Require: Dataset $\mathcal{D}$
Ensure: The learned model $\boldsymbol{w}$.
1: Split data $\mathcal{D}$ into subsets $B_j, j = 1 \dots m$.
2: Initialize $\boldsymbol{\alpha} = \mathbf{0}$.
3: Set cache set $\Omega^{1,1} = \emptyset$.
4: **for** $t = 1, 2, \dots$ (outer iteration) **do**
5: **for** $j = 1, \dots, m$ (inner iteration) **do**
6: Read $B_j = (\boldsymbol{x}_i, y_i)_{i=1\dots m_D}$ from disk.
7: Obtain $d^{t,j}$ by exactly or loosely solving (3.1) on $W^{t,j} = B_j \cup \Omega^{t,j}$, where $\Omega^{t,j}$ is cache.
8: Update $\boldsymbol{\alpha}$ by (3.2).
9: *Select cached data $\Omega^{t,j+1}$ for the next round from $W^{t,j}$.*
10: **end for**
11: **end for**

Instead, we propose a selective block minimization algorithm. On one hand, as we show, our method maintains the convergence property of block minimization by loading a block of new data B_j from disk at each iteration. On the other hand, it maintains a cache set $\Omega^{t,j}$, such that informative samples are kept in memory. At each inner iteration, SBM solves the following sub-problem in the dual on $W^{t,j} = B_j \cup \Omega^{t,j}$:

$$\boldsymbol{d}^{t,j} = \arg\min_{\boldsymbol{d}} \quad f(\boldsymbol{\alpha} + \boldsymbol{d}) \quad \text{s.t.} \quad \boldsymbol{d}_{\bar{W}^{t,j}} = 0, 0 \leq \alpha_i + d_i \leq C, \forall i, \tag{3.1}$$

where $\bar{W}^{t,j} = \{1 \dots l\} \setminus W^{t,j}$ and $f(\cdot)$ is the objective function of the dual SVM problem (Eq. (2.5)). Then it updates the model by solving a sub-problem

$$\boldsymbol{\alpha}^{t,j+1} = \boldsymbol{\alpha}^{t,j} + \boldsymbol{d}^{t,j}. \tag{3.2}$$

In other words, it updates α_W, while fixing $\alpha_{\bar{W}}$. By maintaining a temporary vector $\boldsymbol{w} \in R^n$, solving equation (3.1) only involves the training samples in $W^{t,j}$. After updating the model, SBM selects samples from $W^{t,j}$ to the cache $\Omega^{t,j+1}$ for the next round. Note that the samples in cache $\Omega^{t,j+1}$ are only chosen from $W^{t,j}$ and are already in memory. If a sample is important, though, it may stay in the cache for many rounds, as $\Omega^{t,j} \subset W^{t,j}$. Algorithm 2 summarizes the procedure.

We assume here that the cache size and the data partition are fixed. One can adjust the ratio between $|\Omega|$ and $|W|$ during the training procedure. However, if the partition varies during training, we lose the advantage of storing B_j in a compressed file. In the rest of this section, we discuss the design details of the selective block minimization method.

Solving using Dual Coordinate Decent. In Algorithm 2, a cached sample appears in several sub-problems within an outer iteration. Therefore, primal methods cannot directly apply to our model. In the dual, on the other hand, each α_i corresponds to a sample. Therefore, it allows us to put emphasis on informative samples, and train the corresponding α_i.

We use LIBLINEAR (Fan et al., 2008) as the solver for the sub-problem. LIBLINEAR implements a dual coordinate descent method (Hsieh et al., 2008), which iteratively updates variables in $W^{t,j}$ to solve (3.1) until the stopping condition is satisfied. The following theorem shows the convergence of $\{\boldsymbol{\alpha}^{t,j}\}$ in the SBM algorithm.

Theorem 1. *Assume that a coordinate descent method is applied to solving* (3.1)*. For each inner iteration* (t, j) *assume that*

- *the number of updates,* $\{t^{t,j}\}$*, used in solving the sub-problem is uniformly bounded, and*
- *each* $\alpha_i, \forall i \in W^{t,j}$*, is updated at least once.*

Then, the vector $\{\boldsymbol{\alpha}^{t,j}\}$ *generated by SBM globally converges to an optimal solution* $\boldsymbol{\alpha}^*$ *at a linear rate or better. That is, there exist* $0 < \mu < 1$ *and* k_0 *such that*

$$f(\boldsymbol{\alpha}^{k+1}) - f(\boldsymbol{\alpha}^*) \leq \mu \left(f(\boldsymbol{\alpha}^k) - f(\boldsymbol{\alpha}^*) \right), \quad \forall k \geq k_0.$$

The proof can be modified from (Yu et al., 2010, Theorem 1).

Yu et al. (2010) show that the sub-problem can be solved tightly until the gradient-based stopping condition holds, or solved loosely by going through the samples in $W^{t,j}$ a fixed number of rounds. Either way, the first condition in Theorem 1 holds. Moreover, as LIBLINEAR goes through all the data at least once at its first iteration, the second condition is satisfied too. Therefore, Theorem 1 implies the convergence of SBM.Notice that Theorem 1 holds regardless of the cache selection strategy used.

Selecting Cached Data. The idea of training on only informative samples appears in the selective sampling literature (Loosli et al., 2007), in active set methods for non-linear SVM, and in shrinking techniques in optimization (Joachims, 1999). In the following we discuss our strategy for selecting $\Omega^{t,j}$ from $W^{t,j}$.

During the training, some α_is may stay on the boundary ($\alpha_i = 0$ or $\alpha_i = C$) until the end of the optimization process. Shrinking techniques consider fixing such α_is, thus solving a problem with only unbounded support vectors. We first show in the following theorem that, if we have an oracle of support vectors, solving a sub-problem using only the support vectors provides the same result as if training is done on the entire data.

Theorem 2. *Let* $\boldsymbol{\alpha}^*$ *be the optimal solution of* (2.5)*,* $V = \{i \mid \alpha_i^* = 0\}$ *and* $\bar{V} = \{1 \dots l\} \setminus V$*. If*

$\boldsymbol{\alpha}'$ is the optimal solution of the following problem:

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha}$$

$$\textit{subject to } \boldsymbol{\alpha}_{\bar{V}} = 0, 0 \le \alpha_i \le C, i = 1, \ldots, l,$$

then $f(\boldsymbol{\alpha}') = f(\boldsymbol{\alpha}^*)$ and $\boldsymbol{\alpha}'$ is an optimal solution of (2.5).

The proof is in Appendix A.2. Similarly, if we know in advance that $\alpha_i^* = C$, we can solve a smaller problem by fixing $\alpha_i = C$. Therefore, those unbounded support vectors are more important and should be kept in memory. However, we do not know the unbounded support vectors before we solve (2.5). Therefore, we consider caching unbounded variables ($0 < \alpha_i < C$), which are more likely to eventually be unbounded support vectors.

In dual coordinate descent, α_i is updated in the direction of $-\nabla_i f(\boldsymbol{\alpha})$. Therefore, if $\alpha_i = 0$ and $\nabla_i f(\boldsymbol{\alpha}) > 0$ or if $\alpha_i = C$, and $\nabla_i f(\boldsymbol{\alpha}) < 0$, then α_i is updated toward the boundary. Theorem 2 in (Hsieh et al., 2008) implies such α_i will probably stay on the boundary until convergence.

Based on the shrinking strategy proposed in (Hsieh et al., 2008), we define the following score to indicate such situations:

$$g(\alpha_i) = \begin{cases} -\nabla_i f(\boldsymbol{\alpha}) & \alpha_i = 0, \nabla_i f(\boldsymbol{\alpha}) > 0, \\ \nabla_i f(\boldsymbol{\alpha}) & \alpha_i = C, \nabla_i f(\boldsymbol{\alpha}) < 0, \\ 0 & \text{Otherwise.} \end{cases}$$

$g(\alpha_i)$ is a non-positive score with the property that the lower its value is, the higher the probability is that the corresponding α_i stays on the boundary. We calculate $g(\alpha_i)$ for each sample i, and select the l_c samples with the highest scores into $\Omega^{t,j+1}$, where l_c is the cache size.

In some situations, there are many unbounded variables, so we further select those with the higher gradient value:

$$g(\alpha_i) = \begin{cases} -\nabla_i f(\alpha) & \alpha_i = 0, \\ \nabla_i f(\alpha) & \alpha_i = C, \\ |\nabla_i f(\alpha)| & \text{Otherwise.} \end{cases}$$

To our knowledge, this is the first work that applies a shrinking strategy at the disk level. Our experimental results indicate that our selection strategy significantly improves the performance of a block minimization framework.

3.2 Multi-class Classification

Multi-class classification is important in a large number of applications. A multi-class problem can be solved by either training several binary classifiers or by directly optimizing a multi-class formulation (Har-Peled et al., 2003). One-versus-all takes the first approach and trains a classifier $\boldsymbol{w}^i$ for each class i to separate it from the union of the other labels. To minimize disk access, Yu et al. (2010) propose an implementation that updates $\boldsymbol{w}^1, \boldsymbol{w}^2 \dots \boldsymbol{w}^k$ simultaneously when B_j is loaded into memory. With a similar technique, Algorithm 2 can be adapted to multi-class problems and maintain separate caches $\Omega_u, u = 1 \dots k$ for each sub-problem. In one-versus-all, the k binary classifiers are trained independently and the training process can be easily parallelized in a multi-core machine. However, in this case there is no information shared among the k binary classifiers so the solver has little global information when selecting the cache. In the following, we investigate an approach that applies SBM in solving a direct multi-class formulation (Crammer and Singer, 2000) in the dual:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & f_M(\boldsymbol{\alpha}) = 1/2 \sum_{u=1}^{k} \|\boldsymbol{w}^u\|^2 + \sum_{i=1}^{l} \sum_{u=1}^{k} e_i^u \alpha_i^u \\ \text{subject to} \quad & \sum_{u=1}^{k} \alpha_i^u = 0, \quad \forall i = 1, \dots, l \\ & \alpha_i^u \le C_{y_i}^u, \quad \forall i = 1, \dots, l, u = 1, \dots, k, \text{ where} \end{aligned} \tag{3.3}$$

$$\boldsymbol{w}^u = \sum_{i=1}^{l} \alpha_i^u \boldsymbol{x}_i, \quad C_{y_i}^u = \begin{cases} 0 & \text{if } y_i \neq u, \\ C & \text{if } y_i = u, \end{cases} \quad e_i^u = \begin{cases} 1 & \text{if } y_i \neq u, \\ 0 & \text{if } y_i = u. \end{cases}$$

The decision function is

$$\arg \max_{u=1,\dots,k} (\boldsymbol{w}^u)^T \boldsymbol{x}.$$

The superscript u denotes the class label, and the underscript i represents the sample index. Each sample corresponds to k variables $\alpha_i^1 \dots \alpha_i^k$, and each $\boldsymbol{w}^u$ corresponds to one class. Similar to Algorithm 2, we first split data into m blocks. Then, at each step, we load a data block B_j and train a sub-problem on $W^{t,j} = B_j \cup \Omega^{t,j}$

$$\begin{aligned} \min_{\boldsymbol{d}} \quad & f_M(\boldsymbol{\alpha} + \boldsymbol{d}) \\ \text{subject to} \quad & \sum_{u=1}^{k} (\alpha_i^u + d_i^u) = 0,\ (\alpha_i^u + d_i^u) \le C_{y_i}^u, \quad \forall i, u \\ & d_i^u = 0,\ \text{if } i \notin W^{t,j}, \end{aligned} \tag{3.4}$$

and update the $\boldsymbol{\alpha}$ by

$$\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + \boldsymbol{d},$$

To apply SBM, we need to solve (3.4) only with the data block $W^{t,j}$. Keerthi et al. (2008) proposed a sequential dual method to solve (3.3). The method sequentially picks a sample i and solves for the corresponding k variables $\alpha_i^u, u = 1, 2, \dots, k$. In their update, i is the only sample needed. Therefore, their methods can be applied in SBM. We use the implementation of (Keerthi et al., 2008) in LIBLINEAR.

In Section 3.1, we mentioned the relationship between shrinking and cache selection. For multi-class SBM, (Fan et al., 2008, Appendix E.5) describes a shrinking technique implemented in LIBLINEAR. If α_i^u is on the boundary (i.e., $\alpha_i^u = C_i^u$) and satisfies certain conditions, then it is shrunken. To select a cache set, we consider a sample 'informative' if a small fraction of its corresponding variables $\alpha_i^u, u = 1, 2, \dots, k$ are shrunken. We keep such samples in the cache $\Omega^{t,j}$. Sec. 3.6 demonstrates the performance of SBM in solving (3.3).

3.3 Learning from Streaming Data

Learning over streaming data is important in many applications when the amount of data is just too large to keep in memory; in these cases, it is difficult to process the data in multiple passes and it is often treated as a stream that the algorithm needs to process in an online manner. However, online

learners make use of a simple update rule, and a single pass over the samples is typically insufficient to obtain a good model. In the following, we introduce a variant of SBM that considers Algorithm 2 with only one iteration. Unlike existing approaches, we accumulate the incoming samples as a block, and solve the optimization problem one data block at a time. This strategy allows SBM to fully utilize its resources (learning time and memory capacity) and to obtain a significantly more accurate model than in standard techniques that load each sample from disk only once.

When SBM collects a block of samples B from the incoming data, it sets the corresponding $\alpha_i, i \in B$ to 0. Then, as in Algorithm 2, it updates the model by solving (3.1) on $W = B \cup \Omega$, where Ω is the cache set collected from the previous step. In Section 3.1, we described SBM as processing the data multiple times to solve Eq. (2.5) exactly. This required maintaining the whole $\boldsymbol{\alpha}$. When the number of instances is large, storing $\boldsymbol{\alpha}$ becomes a problem (see Sec. 3.4.2). However, when dealing with streaming data each sample is loaded once. Therefore, if a sample i is removed from memory, the corresponding α_i will not be updated later, and keeping $\alpha_i, i \in W$ is sufficient. Without the need to store the whole $\boldsymbol{\alpha}$, the solver is able to keep more data in memory.

The SBM for streaming data thus obtains a feasible solution that approximately solves (2.5). Compared to other approaches for streaming data, SBM has two advantages:

- By adjusting the cache size and the tightness of solving the sub-problems, we can obtain models with different accuracy levels. Therefore, the solver can adjust to the available resources.
- As we update the model on both a set of incoming samples and a set of cached samples, the model is more stable than the model learned by other online learners.

We study its performance experimentally in Sec. 3.6.2.

3.4 Implementation Issues

Yu et al. (2010) introduce several techniques to speed up block minimization algorithms. Some techniques such as data compression and loading blocks in a random order can also be applied to Algorithm 2. In the following, we further discuss implementation issues that are relevant to SBM.

3.4.1 Storing Cached Samples in Memory

In Step 4(d) of Algorithm 2, we update the cache set $\Omega^{t,j}$ to $\Omega^{t,j+1}$ by selecting samples from $W^{t,j}$. As memory is limited, non-cached samples $i \notin \Omega^{t,j+1}$ are removed from memory. If data is stored in a continuous array[1] we need a careful design to avoid duplicating samples.

Assume that $W^{t,j}$ is stored in a continuous memory chunk MEM_W. The lower part of MEM_W, MEM_Ω, stores the samples in $\Omega^{t,j}$, and the higher part stores B_j. In the following we discuss a way to update the samples in MEM_Ω from $W^{t,j} = \Omega^{t,j} \cup B_j$ to $W^{t,j+1} = \Omega^{t,j+1} \cup B_{j+1}$ without using extra memory. We want to update MEM_W and store $\Omega^{t,j+1}$ in its lower part. However, as $\Omega^{t,j+1}$ is selected from $\Omega^{t,j} \cup B_j$, which is currently in Mem_W, we need to move the samples in such an order that the values of samples in $\Omega^{t,j+1}$ will not be covered. Moreover, as data is stored in a sparse format, the memory consumption for storing each sample is different. We cannot directly swap the samples in memory. To deal with this problem, we first sort the samples in $\Omega^{t,j+1}$ by their memory location. Then, we sequentially move each sample in $\Omega^{t,j+1}$ to a lower address of MEM_W and cover those non-cached samples. This way, we form a continuous memory chunk $\Omega^{t,j+1}$. Finally, we load B_{j+1} after the last cached sample in $\Omega^{t,j+1}$; MEM_W stores $W^{t,j+1}$ now.

3.4.2 Dealing with a Large $\boldsymbol{\alpha}$

When the number of examples is huge, storing $\boldsymbol{\alpha}$ becomes a problem. This is especially the case in multi-class classification, as the size of $\boldsymbol{\alpha}$ grows rapidly since each sample corresponds to k elements in $\boldsymbol{\alpha}$. In the following we introduce two approaches to cope with this situation.

1. Storing $\alpha_i, i \notin W$ in a Hash Table. In real world applications, the number of support vectors is sometimes much less than the number of samples. In these situations, many α_is will stay at 0 during the whole training process. Therefore, we can store the α_is in a sparse format. That is, we store non-zero $\alpha_i > 0$ in a hash table. We can keep a copy of the current working $\alpha_i, i \in W$ in an array, as accessing data from a hash table is slower than from an array. The hash strategy is widely used in developing large scale learning systems. For example, Vowpal Wabbit uses a hash table to store the model $\boldsymbol{w}$ when n is huge.

2. Storing $\alpha_i, i \notin W$ in Disk. In solving a sub-problem, $\alpha_i, i \notin W^{t,j}$ are fixed. Therefore, we

[1]This operation can be performed easily if the data is stored in a linked list. However, accessing data in a linked list is slower than in an array.

can store them in disk and load them when the corresponding samples $B_j, i \in B_j$ are loaded. We maintain m files to save those α_is which are not involved in the current sub-problem. Then, we save the α_i to the jth file when the corresponding sample $i \in B_j$ is removed from memory. To minimize disk access, the α_is can be stored in a sparse format, and only the α_is with non-zero value are stored.

3.5 Related Work

In Sections 3.1 we discussed the differences between the block minimization framework and our Algorithm 2. In the following, we review other approaches to large scale classification and their relations to Algorithm 2.

Online Learning. Online learning is a popular approach to learning from huge data sets. An online learner iteratively goes over the entire data and updates the model as it goes. A commonly held view was that online learners efficiently deal with large data sets due to their simplicity. In the following, we show that the issues are different when the data does not fit in memory. At each step, an online learner performs a simple update on a single data instance. The update done by an online learner is usually cheaper than the one done by a batch learner (e.g., Newton-type method). Therefore, even though an online learner takes a large number of iterations to converge, it could obtain a reasonable model quickly (Bottou and LeCun, 2004; Shalev-Shwartz and Srebro, 2008). However, when the data cannot fit into memory, we need to access each sample from disk again every round. An online method thus requires large I/O due to the large number of iterations required. One may consider a block of data at a time to reduce the number of iterations.

Some online learning algorithms have considered solving a block of data at a time. For example, Pegasos (Shalev-Shwartz et al., 2007) allows a block-size update when solving (2.4) with projected stochastic gradient descent. However, as they assume data points are randomly drawn from the entire data, they cannot do multiple updates on one block. Therefore, it still cannot fully utilize the data in memory. Some learning methods have considered caching techniques that are similar to our proposal. Guyon et al. (1992) improved the recognition rate by emphasizing the atypical examples in the training data. Collins and Roark (2004) suggested periodically updating the model on samples the model gets wrong. However, their caching techniques are ad-hoc and have

no convergence guarantees. To our knowledge, SBM is the first method that uses caching and *provably converges* to (2.4).

Parallelizing Batch Algorithm. Although parallel algorithms such as (Woodsend and Gondzio, 2009; Chang et al., 2007a) can scale up to huge problems, the communication across machines still incurs a large overhead. Moreover, the aforementioned algorithms still require that training is done on a single machine, so we do not consider parallel solutions in this work. Recently there have been proposals for solving large problems by parallelizing online methods using multi-core machines (Langford et al., 2009b). However, these methods only shorten the learning time and do not solve the I/O problem.

Reducing Data Size. Sub-sampling and feature pruning techniques as in (Yu et al., 2003; Balcázar et al., 2001) reduce the size of the data. In some situations, training on a subset of the data is sufficient to generate a reasonable model. But in other situations, the huge amount of data is already preprocessed and further pruning may degrade the performance. In fact, in SBM we start from a model trained on a randomly drawn data block, and iteratively improve the model by loading other sample blocks. Therefore, its performance should be at least as good as random sub-sampling.

Bagging Models Trained on a Subset of the Data. Bagging (Breiman, 1996) is an algorithm that improves test performance by combining models trained on subsets of data. To train on large data, a bagging algorithm randomly picks k-subsets of the entire data, and trains k models $\{w^1, w^2, \ldots, w^k\}$ separately on each subset. The final model is obtained by averaging the k models. Some analysis was provided recently in (Zinkevich et al., 2010). Although a bagging algorithm can sometimes learn a good model, it does not solve Eq. (2.4). Therefore, in practice, it is not clear if a bagging model can obtain as good a classifier as obtained by solving (2.4).

Selective Sampling. Selective sampling and active learning methods select informative samples for learning a model (Pérez-Cruz et al., 2004; Tong and Koller, 2002). Even though their goal is different, these methods can be applied when training on large data sets (See Sec. 6 in (Yu et al., 2003)). Starting from a subset of the data, selective sampling methods iteratively select the samples that are close to the current margin, and update the model on those samples. To compute the margin accurately, a selective sub-sampling method may require access to the entire data from

disk, which is expensive. In contrast, SBM loads a data block not only for selection but also for training. It selects samples into the cache only if they are already in memory. Therefore, it suffers a very small overhead when selecting samples. Moreover, selective sampling is not guaranteed to converge to (2.4), while SBM converges linearly.

Related Methods for Non-Linear Models. In solving non-linear models, data is usually mapped into a high dimensional feature space via a kernel trick. When the kernel matrix goes beyond memory capacity, an active set method (e.g., (Vishwanathan et al., 2003; Scheinberg, 2006; Bordes et al., 2005)) maintains an active set A and updates only the variables $\alpha_i, i \in A$ each time. If $|A|$ is small, the corresponding kernel entries can be stored in memory and the active set method yields fast training. Active set methods are related to our work as they also select samples during training. However, the memory problem of dealing with non-linear models is different from the one we face. Non-linear solvers usually assume that samples are stored in memory, while the entire kernel cannot be stored. Therefore, there is no restriction on accessing data. In our case, the samples can only be loaded sequentially from disk.

Another algorithm, Forgetron (Dekel et al., 2008), is related to our work. However, the focus there is different. When training a model with a non-linear kernel, the model is represented by some samples (support vectors). When they have too many samples to store in memory, Forgetron selects the samples which can represent the current model best. In the linear case, $\boldsymbol{w}$ can be directly stored. In our case, on the other hand, the selection process is done in order to speed up the training rather than to just represent the model more concisely.

3.6 Experiments

In this section, we first show that SBM provides an improvement over the block minimization algorithm (BMD) Yu et al. (2010). Then, we demonstrate that SBM converges faster than an online learner. Finally, we investigate the setting in which each sample is loaded from disk only once and show that SBM deals better with streaming data.

We consider two large binary classification data sets, webspam and kddcup10, and one multi-class data set, prep. webspam is a document collection; kddcup10 is taken from educational applications

Data set	l	l_t	n	#nonzeros	Memory	C	#nBSV	#classes
webspam	280k	70k	16,609k	1,043m	16.7GB	64	7k	2
kddcup10	19,264k	748k	29,890k	566m	9GB	0.1	2,775k	2
prep	60,000k	5,710k	11,148k	918m	14GB	0.1	-	10

Table 3.1: Data statistics: We assume the data is stored in a sparse representation. Each feature with non-zero value takes 16 bytes on a 64-bit machine due to data structure alignment. l, n, and #nonzeros are numbers of instances, features, and non-zero values in each training set, respectively. l_t is the number of instances in the test set. Memory is the memory consumption needed to store the entire data. #nBSV shows the number of unbounded support vectors ($0 < \alpha_i < C$) in the optimal solution. #classes is the number of class labels.

and has been used in the data mining challenge.[2] prep is a 10-classes classification task, aiming at predicting the correct preposition in an English sentence with a bag-of-words model (Rozovskaya and Roth, 2010).[3] In kddcup10 and webspam, the feature vectors are scaled to unit length, so that $\forall i, \|\boldsymbol{x}_i\| = 1$. prep takes binary features, and we prune the features which appear less than 5 times in all samples. Table 3.1 lists the data statistics. As shown in (Yu et al., 2010), applying sub-sampling or bagging downgrades the test performance on kddcup10 and webspam. Each data set is stored as an ASCII file before the training process.

We tune the penalty parameter C using a five-fold cross-validation on the training data and report the performance of solving (2.5) with the best parameter. This setting allows SBM for linear SVM to be compared with other learning packages such as VW. The implementation of BMD and SBM is in C/C++[4] with double precision in a 64-bit machine. We restrict each process to use at most 2GB memory.

3.6.1 Training Time and Test Accuracy

Comparison between SBM and BMD. We compare SBM with the block minimization algorithm in the dual (BMD) (Yu et al., 2010), as both the algorithms consider a block of data at a time. For a fair comparison, we take the largest block size ($|D_{\text{BM}}| = \max_j |B_j|$) in BMD as a reference, and adjust SBM to use the same amount of memory during training. We consider three variants

[2] kddcup10 is preprocessed from the second data set in KDD Cup 2010. webspam and kddcup10 can be downloaded at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`.

[3] prep is extracted from Wikipedia and New York Times. As the annotation can be automatically generated from articles (assuming the given text is correct), the supervised data can be very large.

[4] The experimental code is modified from Yu et al. (2010) `http://www.csie.ntu.edu.tw/~cjlin/liblinear/exp.html`.

of Algorithm 2: SBM-1/2, SBM-1/3, and SBM-1/2-rand. SBM-1/2 and SBM-1/2-rand load half the memory with new data and the rest is kept for the cache ($|B_j| = |\Omega| = 1/2|D_{\mathrm{BM}}|$). SBM-1/3 reserves 1/3 of the memory capacity for cache and loads the rest with new data ($|\Omega| = 1/3|D_{\mathrm{BM}}|$ and $|B_j| = 2|\Omega|$). Both SBM-1/2 and SBM-1/3 select the cached sample using the gradient information as described in Sections 3.1 and 3.2. SBM-1/2-rand selects the cached sample by picking randomly from the samples in memory (W). All sub-problems in BMD and SBM-* are solved by a dual coordinate method with 10 rounds ($t^{i,j} = 10$). We split both webspam and kddcup10 into 12 blocks ($m = 12$)[5] and split prep into 15 blocks in BMD. The cache size and block size of SBM-* are adjusted according to BMD. Each block is stored in a compressed file. The compression introduces additional cost of storing compressed files, but it reduces the load time of each block (see (Yu et al., 2010, Sec. 5.1)). We report the *wall clock* time in all experiments including the initial time to generate the compressed files (the time to read data from an ASCII file, split it, and store the compressed data in disk.)

The number of α_i^us in prep is 600 million. Therefore, storing all α_i^u takes 4.8GB memory which exceeds the memory restriction. Following the discussion in Sec. 3.4.2, we store the unused α_i^u in files for BMD and SBM-*.

To check the convergence speed of each method, we compare the relative difference between the dual objective function value (2.5) and the optimum ($f(\boldsymbol{\alpha}^*)$),

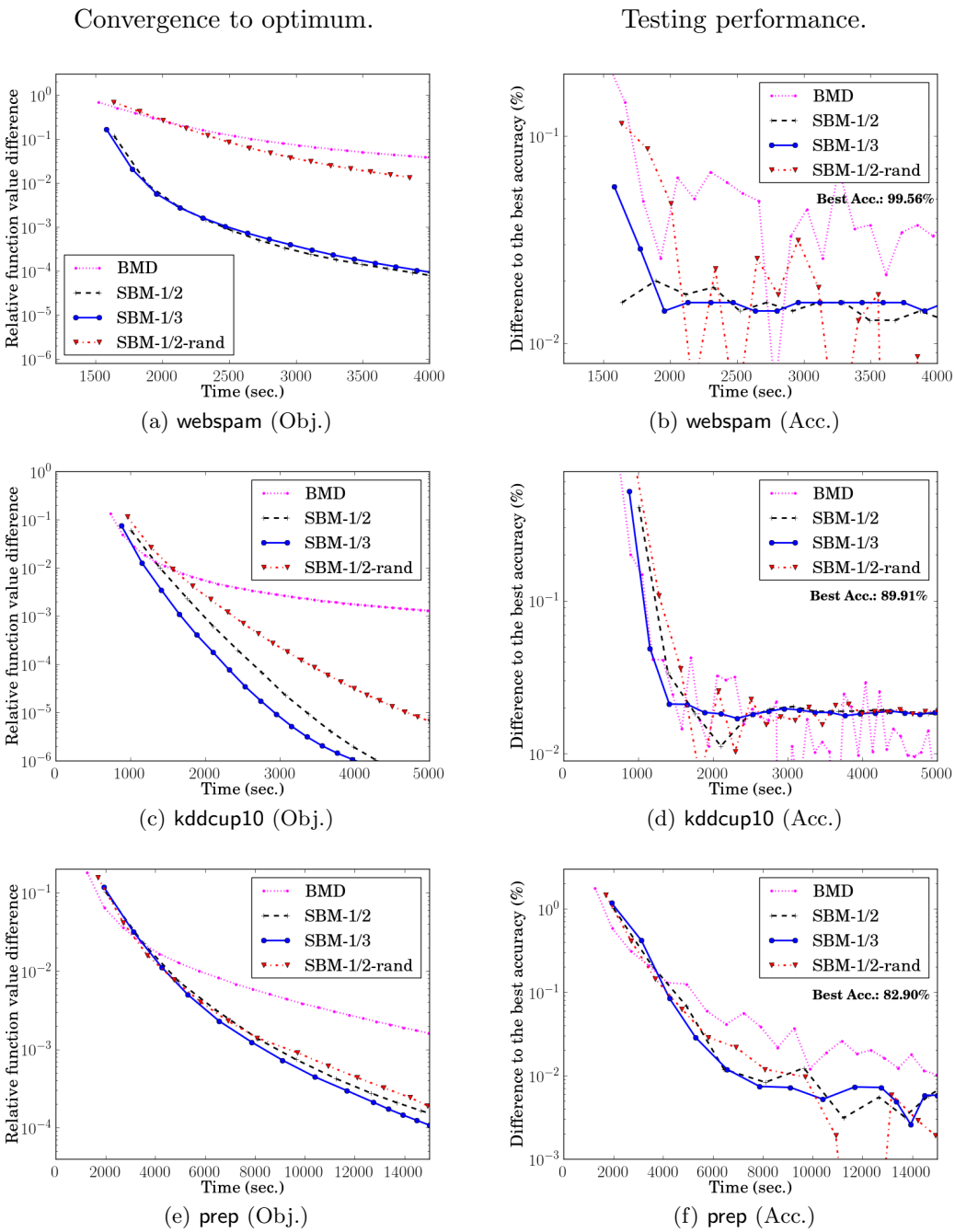
$$|(f(\boldsymbol{\alpha}) - f(\boldsymbol{\alpha}^*))/f(\boldsymbol{\alpha})|\,, \tag{3.5}$$

over time. We are also interested in the time it takes each method to obtain a reasonable and stable model. Therefore, we compare the difference between the performance of the current model ($\mathrm{acc}(\boldsymbol{w})$) and the best test accuracy among all methods (acc^*)

$$(\mathrm{acc}^* - \mathrm{acc}(\boldsymbol{w}))\%. \tag{3.6}$$

Note that tightly solving (2.4) is not guaranteed to have the best test performance. Some solvers may reach a higher accuracy before the model converges.

[5] Although the data size of webspam is larger than kddcup10. kddcup10 has a huge number of samples and features, and it takes about 500MB to store the model.



(a) webspam (Obj.) (b) webspam (Acc.)

(c) kddcup10 (Obj.) (d) kddcup10 (Acc.)

(e) prep (Obj.) (f) prep (Acc.)

Figure 3.1: Comparison between the proposed SBM-* and BMD (Yu et al., 2010). Left: the relative function value difference (3.5) to the minimum of (2.5). Right: the test accuracy difference (3.6) from the best achievable accuracy. Time is in seconds, and the y-axis is in log-scale. Each marker indicates one outer iteration. As all the solvers access all samples from disk once at each outer iteration, the markers indicate the amount of disk access.

Figure 3.1 shows that all the SBM-* outperform BMD significantly. By selecting cached samples using the gradient information, our key methods, SBM-1/2 and SBM-1/3 perform even better. In

particular, when the unbounded support vectors can fit into memory, SBM-1/2 and SBM-1/3 keep most of them in cache during the training process and thus enjoy fast convergence.[6] Consequently, the models generated are more stable in test. The performance of SBM-1/2 and SBM-1/3 are similar. This indicates that the model is not sensitive to the ratio between block size and cache size. Note that SBM is more general than BMD, as it is reduced to BMD when the cache size is zero.

It is interesting to note that even when the cached set is selected randomly, SBM-1/2-rand is still faster than BMD. One possible reason is that partitioning the data affects the performance of the solver. BMD uses a fixed partition $B_j, \forall j$, throughout the training process. For SBM, although, $\forall j, B_j$ is fixed as well, the cached set Ω varies at each iteration. Therefore, SBM avoids wasting time on a bad data block. When the number of bounded support vectors is large, the selection of Ω is less important since drawing a sample randomly already has a high probability of including bounded support vectors in the cache (see figure 3.1c). SBM-* works better in binary classification than in multi-class classification, as sub-problems of the multi-class formulation take more learning time. Thus, the I/O overhead plays a smaller role at training time.

Comparison between SBM and Online Learners. In Sec. 3.5, we will discuss that people in the community believe that online learners are fast for large data sets since they make use of a simple update rule. However, as we show, online learners use more I/O time when the data is not in memory. We compare the following solvers:

- VW[7]: an online learning package (version 5.1).[8] It implements a stochastic gradient descent method (Freund et al., 1997).
- BM-Pegasos: A version of Pegasos implemented under a block minimization framework (Yu et al., 2010; Shalev-Shwartz et al., 2007).
- BMD-in-1: Similar to BMD but the inner solver goes through the samples in each sub-problem only once.
- SBM: The same as SBM-1/2 in the previous experiment.

We implemented Pegasos under the block minimization framework such that it saves considerable

[6] E.g., in figure 3.1a, more than 98% of the unbounded support vectors are stored in cache after 2 iterations.

[7] `https://github.com/JohnLangford/vowpal_wabbit`

[8] We use the latest version 5.1, and set the parameter as -b 24 --loss_function hinge --initial_t 80000 --power_t 0.5 -l 1000 --compressed as the author suggested.

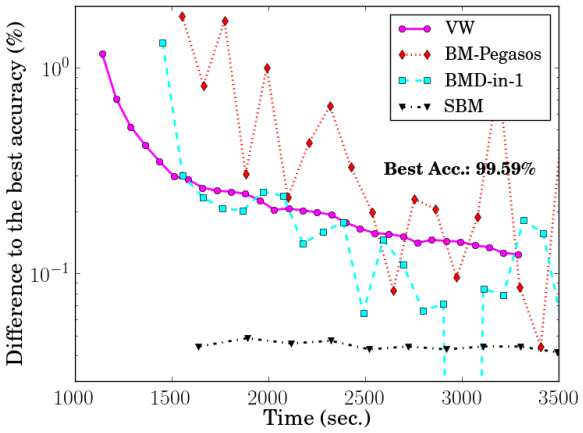


Figure 3.2: A comparison between SBM and online learning algorithms on webspam. We show the difference between test performance and the best accuracy achievable (3.6). Time is in seconds. The markers indicate the time it takes each solver to access data from disk once. Except for VW, each solver takes about 90 sec. in each round to load data from the compressed files.

I/O time by loading data from a compressed file. Although VW also considers a hinge loss function, it does not converge to (2.5). Therefore, its final accuracy is different from that of the others. Except for SBM, all the methods update the model on each sample once per outer iteration. BMD-in-1 takes the same update rule as the passive-aggressive online algorithm (Crammer et al., 2006), but it keeps $\boldsymbol{\alpha}$ and its solution converges to (2.5). Except for VW, all methods take about 90 seconds to access each sample from the compressed files. VW has a faster implementation[9] for loading data and takes about 70 seconds per outer iteration.

Figure 3.2 shows the test performance over time (3.6) on webspam. SBM converges faster than other online methods to the final performance. *The reason is that an online learner spends most of its time loading data instead of learning a model (see Sec. 3.5).* For example, BM-Pegasos spends only 15 seconds updating the model. Therefore, it spends 85% of the training time loading data. In contrast, the learning time per iteration in SBM is 180 seconds. Thus, SBM uses 2/3 of the training time to update the model and consequently converges faster.

[9] VW uses single floating point precision while our implementations use double precision. Although using single precision reduces the training time, numerical inaccuracy may occur. In addition, VW is a multi-threaded program. It uses one thread for loading data and another thread for learning.

Method	webspam				kddcup10			
	Training Time (sec.)			Acc.	Training Time (sec.)			Acc.
	I/O	Learning	Total		I/O	Learning	Total	
Perceptron	5078	31	5108	98.66	1403	42	1448	84.93
MIRA	5392	27	5419	98.73	1265	44	1312	86.14
CW	4698	59	4757	99.49	1439	99	1535	89.07
VW	-	-	507	98.42	-	-	223	85.25
BM-Pegasos	470	9	479	97.87	187	20	207	87.48
BMD	472	60	532	99.48	181	201	382	88.62
SBM-in-3	484	81	565	**99.55**	181	151	332	89.54
SBM	473	177	650	**99.55**	180	442	634	**89.67**
Reference	-	-	-	99.55	-	-	-	89.90

Method	prep			
	Training Time (sec.)			Acc.
	I/O	Learning	Total	
Perceptron	1843	903	2753	76.10
MIRA	1829	816	2652	76.81
CW	1820	1121	2949	77.70
BMD	265	649	914	80.06
SBM-in-3	321	778	1099	81.61
SBM	354	1956	2310	**81.80**
Reference	-	-	-	82.90

Table 3.2: Training time (*wall clock time*) and test accuracy for each solver *when each sample is loaded into memory only once*. We show both the loading time (the time to read data from an ASCII file) and the learning time. The reference model is obtained by solving (2.4) until it converges (see text). We show that by updating the model on one data block at a time, SBM obtains an accurate model for streaming data. Note that the time reported here is different from the time required to finish the first iteration in Figure 3.1 (see text). Some entries in the table are missing. BM-Pegasos and VW do not support multi-class classification. Also, it is difficult to access the I/O and learning time of VW since it is a multi-threaded program. The top three methods are implemented in Java, and the rest are in C/C++.

3.6.2 Experiments on Streaming Data

In Sec. 3.3, we introduced a variation of SBM that deals with streaming data. Now we investigate its performance.

In addition to the solvers BMD, SBM, BM-Pegasos and VW introduced in previous experiments, we further compare them to the implementations in (Crammer et al., 2009).[10] The package

[10] `http://www.cs.jhu.edu/~mdredze/`. The original package takes a very long time to train on the webspam data

currently supports the following solvers for data streams:

- MIRA: an online learning algorithm proposed in (Crammer et al., 2006).
- CW: a confidences weighted online algorithm (Dredze et al., 2008; Crammer et al., 2009).
- Perceptron: an online learning algorithm.

In addition, we consider a setting of SBM, SBM-in-3, in which we only allow 3 rounds when solving each sub-problem (the general setting allows 10 rounds). The training time of this version of SBM is similar to the training time of BMD. Note that MIRA, CW and Perceptron are implemented in JAVA, and the rest of the solvers are implemented in C/C++. For multi-class problems, we used the top-1 approach (Crammer et al., 2009) in MIRA, CW and Perceptron.

Table 3.2 lists the training time and test accuracy when each sample is loaded from disk only once. To see if loading data once can achieve a result similar to running over the data multiple times, we show a reference model in the table, which solves Eq. (2.4) until the model converges. We provide both the time to load data from an ASCII file (I/O) and the update time of the model (Learning). For each solver, we focus on the ratio of loading to learning time rather than on comparing the absolute times across different solvers. Note that the time reported in Table 3.2 is different from the time to complete the first round in Figure 3.1. Figure 3.1 includes the time it takes to generate compressed files from an ASCII file. However, for streaming data, each sample is loaded into memory once and there is no need to store data in compressed files, resulting in shorter I/O time.

The results show that SBM is able to utilize the memory capacity and spend more learning time to achieve an accurate model. It even achieves a model that is almost as accurate as the reference model on webspam. As the I/O is expensive, for all the methods, the loading time takes a large fraction of the total training time. Online methods such as MIRA and Perceptron use a simple update rule. Their learning time is indeed small, but the model learned with a single pass over the samples is far from converging. CW uses a better update rule and achieves a more accurate model, but the performance is still limited. In contrast, both SBM and BMD are able to update on a data block at a time, thus taking more information into account. Therefore, they achieve a better result. Comparing SBM-in-3 to BMD we see that SBM-in-3 has a similar training time to that of BMD,

set. It might be due to the deep class hierarchy in the package. We fixed the problem and sped up the package.

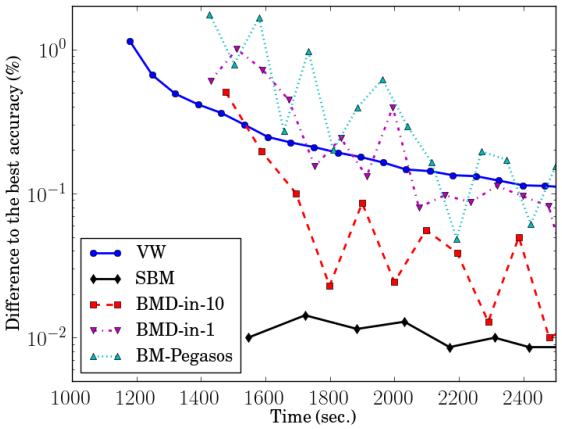


Figure 3.3: A comparison among SBM, BMD and online learning algorithms on webspam. All the methods are implemented with single precision. We show the difference between test performance and the best accuracy achievable (3.6). Time is in seconds. The markers indicate the time it takes each solver to access data from disk once.

but learns a model with better accuracy. As shown, by keeping informative samples in memory, SBM utilizes the memory storage better. When the resources are limited, users can adjust the test performance of SBM by adjusting the memory usage and the accuracy level of solving the sub-problems.

3.6.3 Experiments Under The Single Precision Floating Point Arithmetic

Some packages (e.g., VW) consider storing floating point values in single precision. Using single precision has the following advantages:

- Each feature with non-zero value needs only 8 bytes. Therefore, it takes less amount of memory to store a sample.
- Calculations with single precision are usually faster than with double precision.

However, using single precision sometimes induces numerical inaccuracy. For the sake of robustness, some packages (e.g., LIBLINEAR) use double precision to store the data and the model.

In Section 3.6.1, we follow the setting in (Yu et al., 2010) to implement SBM and BM-* in double precision and keep VW using single precision. Therefore, Figure 3.2 shows that the time cost per iteration of VW is smaller than the other methods implemented in double precision. In

this section, we study the test performance along time of each method under the single precision arithmetic on the webspam data set. Besides the solvers showed in Figure 3.2, we include a dual block minimization method that solving the sub-problem with 10 rounds (BMD-in-10). Figure 3.3 shows the results. By using single precision, all the methods under the block minimization framework (SBM and BM-*) reduce about 25% training time. In the figure, VW, BM-Pegasos and BMD-in-1 take small effort at each iteration, but they require considerable number of iterations to achieve a reasonable model. In contrast, the cost of BMD-in-10 and SBM per iteration is high, but they converge faster.

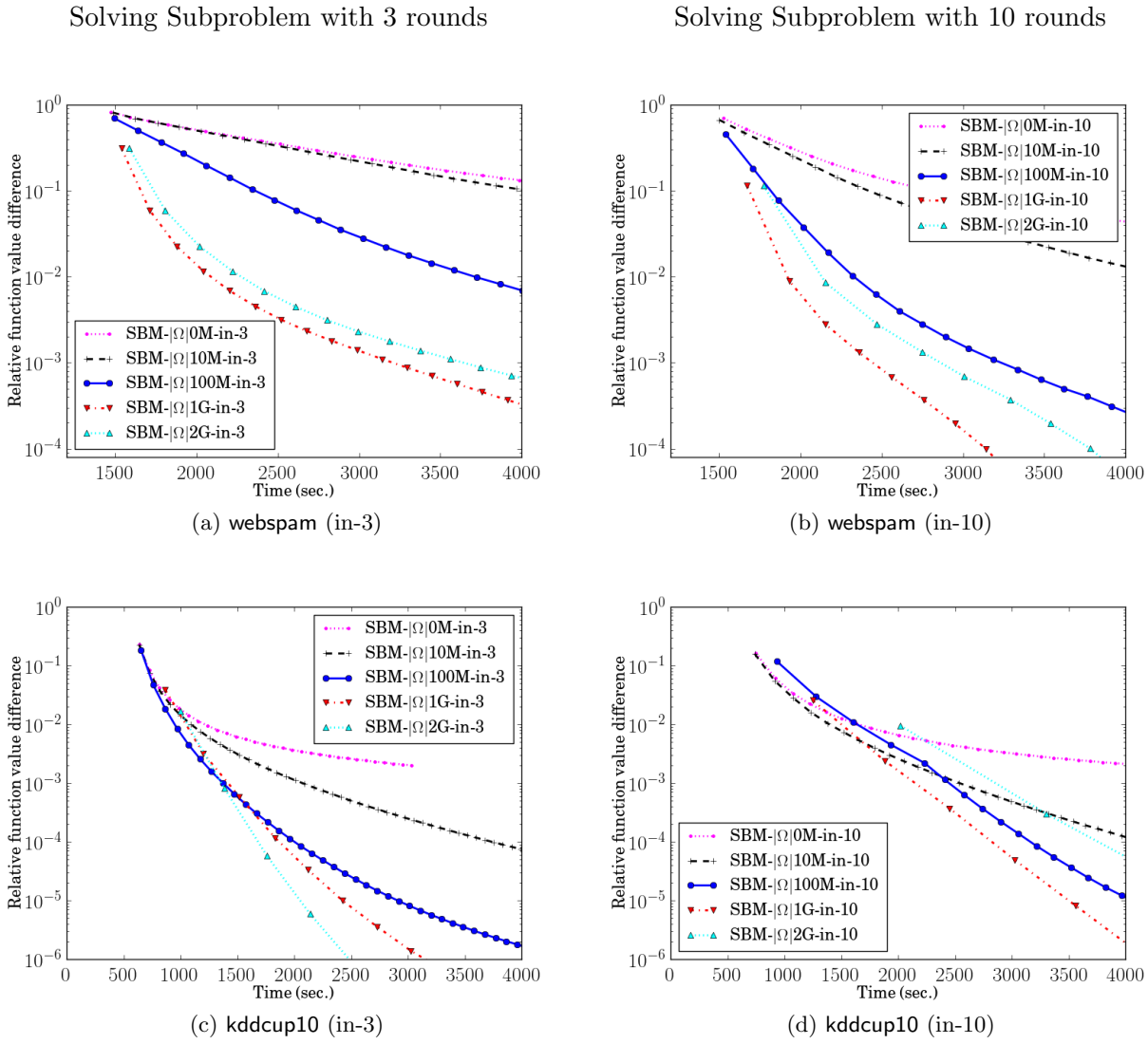
3.6.4 Cache Size

In this section, we investigate how the cache size affects the performance of SBM. We experiment on the two binary classification problems, webspam and kddcup10. We set the cache size of SBM to 0B (equivalent to BMD), 10MB, 100MB, 1GB and 2GB. We demonstrate the results of SBM when the sub-problem is solved with 3 rounds and 10 rounds, respectively.

Figure 3.4 shows the relative dual function value difference to the optimum along time (3.5). In general, large cache size yields faster convergence. The reason is that when the cache size increases, informative samples are more likely to stay in memory. Therefore, SBM takes advantage of updating model using more important samples. However, enlarging cache size may not always improve the performance. For example, in Figure 3.4d, SBM-$|\Omega|$1G-in-10 is faster than SBM-$|\Omega|$2G-in-10. The reason seems to be that when the cache is large enough to store all the informative samples, SBM starts to cache unimportant samples in memory. As a result, it wastes time to put effort on those unimportant samples. How to choose the cache size and the inner stopping condition of SBM properly is a further research issue.

3.7 Conclusion

In summary, we presented a selective block minimization algorithm for training large-scale linear classifiers under memory restrictions. We also analyzed the proposed algorithm and provided extensive experimental results. To reduce the amount of disk access, we suggested caching informative samples in memory and updating the model on the important data more frequently. We show that,



(a) webspam (in-3)

(b) webspam (in-10)

(c) kddcup10 (in-3)

(d) kddcup10 (in-10)

Figure 3.4: The relative function value difference (3.5) to the optimum of (2.5). Time is in seconds, and the y-axis is in log-scaled. We show the performance of SBM with various cache sizes. SBM-|Ω|0M-in-* is equivalent to BMD. We demonstrate both the situations that SBM solves the sub-problem with 3 rounds and 10 rounds.

even though the proposed method treats data points non-uniformly, it provably converges to the global optimum on the entire data. Our experimental results on binary, multi-class and streaming data, show the significant advantage of SBM over block minimization and online algorithms. SBM converges quicker to a more accurate and stable model.

Chapter 4

Multi-core Structural SVM Training

In Chapter 2.2.2, we introduced a dual coordinate descent method for solving structured SVM (i.e., Eq. (2.14)). As we mentioned, the number of dual variables in Eq. (2.14) is exponentially large. Since most dual variables stay at 0 from the beginning until the end of the optimization process, we can maintain an active set $\mathcal{A}$ to select dual variables that are likely to be non-zero in the final model, and update the model using only these variables. However, this approach is inherently single-threaded, and extending it to a multi-core environment is not trivial. Consequently, it cannot take advantage of the multiple cores available in most modern workstations.

Existing parallel algorithms for training structured SVMs (such as (Chang et al., 2010)) use a sequence of two phases: an inference phase, where loss-augmented inference is performed using the current model, and a model update phase. Each phase is separated from the other by a barrier that prevents model update until the inference is complete and vice versa. A similar barrier exists in map-reduce implementations of the binary SVM (Chu et al., 2007) and the Perceptron (McDonald et al., 2010a) algorithms. Such barrier-based approaches prevent existing parallel algorithms from fully utilizing the available processors.

In (Chang et al., 2013b), we propose the DEMI-DCD algorithm (DEcoupled Model-update and Inference with Dual Coordinate Descent), a new parallel algorithm based on the dual coordinate descent (DCD) method for the L2-loss structured SVM. DCD has been shown competitive with other optimization techniques such as the cutting plane method (Tsochantaridis et al., 2005) and the Frank-Wolfe method (Lacoste-Julien et al., 2013). DEMI-DCD removes the need for a barrier between the model update and candidate structure selection phases allowing us to distribute these two steps across multiple cores. We show that our approach has the following advantages:

1. DEMI-DCD requires little synchronization between threads. Therefore, it fully utilizes the

computational power of multiple cores to reduce training time.

2. As in the standard dual coordinate descent approach, DEMI-DCD can make multiple updates on the structures discovered by the loss-augmented inference, thus fully utilizing the available information. Furthermore, our approach retains the convergence properties of dual coordinate descent.

We evaluate our method on two NLP applications – part-of-speech tagging and entity-relation extraction from text. In both cases, we demonstrate that not only does DEMI-DCD converge faster than existing methods to better performing solutions (according to both primal objective value and test set performance), it also fully takes advantage of all available processors unlike the other methods. For the part-of-speech tagging task, we show that with 16 threads, our approach reaches a relative duality gap of 1% in 192 seconds, while a standard multi-threaded implementation of the dual coordinate descent algorithm with 16 threads takes more than 600 seconds to reach an equivalent solution. Similarly, for the entity-relations task, our approach reaches within 1% of the optimal within 86 seconds, compared to 275 seconds for the baseline.

4.1 A Parallel Algorithm for Training Structural SVM Models

In this section, we describe the parallel learning algorithm DEMI-DCD for solving Eq. (2.13) which decouples the model update steps from the inference steps during learning, and hence fully utilizes the computational power of multi-core machines.

Let p be the number of threads allocated for learning. DEMI-DCD first splits the training data $\mathcal{D}$ into $p-1$ disjoint parts: $\{B_j\}_{j=1}^{p-1}$ with each $B_j \subset \mathcal{D}$. Then, it generates two types of threads, a learning thread and $p-1$ active set selection threads. The j^{th} active set selection thread is responsible for maintaining an active set $\mathcal{A}_i$ for each example i in the part B_j. It does so by searching for candidate structures for each instance in B_j using the current model $\boldsymbol{w}$ which is maintained by the learning thread.

The learning thread loops over all the examples and updates the model $\boldsymbol{w}$ using $\alpha_{i,\boldsymbol{y}} \in \mathcal{A}_i$ for the i^{th} example. The $p-1$ active set selection threads are independent of each other, and they share $\mathcal{A}_i$ and $\boldsymbol{w}$ with the learning thread using shared memory buffers. We will now discuss our

model in detail. The algorithms executing the learning and the active set selection threads are listed as Algorithm 3 and 4 respectively.

Learning Thread. The learning thread performs a two-level iterative procedure until the stopping conditions are satisfied. It first initializes $\boldsymbol{\alpha}^0$ to a zero vector, then it generates a sequence of solutions $\{\boldsymbol{\alpha}^0, \boldsymbol{\alpha}^1, \ldots\}$. We refer to the step from $\boldsymbol{\alpha}^t$ to $\boldsymbol{\alpha}^{t+1}$ as the outer iteration. Within this iteration, the learning thread sequentially visits each instance $\boldsymbol{x}_i$ in the data set and updates $\alpha_{i,\boldsymbol{y}} \in \mathcal{A}_i$ while all the other dual variables are kept fixed. To update $\alpha_{i,\boldsymbol{y}}$, it solves the following one variable sub-problem that uses the definition of the dual objective function from Eq. (2.14):

$$\begin{aligned} \bar{d}_{i,\boldsymbol{y}} &= \arg\min_d D(\boldsymbol{\alpha} + \boldsymbol{e}_{i,\boldsymbol{y}} d) \quad \text{s.t.} \quad d\alpha_{i,\boldsymbol{y}} + d \geq 0 \\ &= \arg\min_d \frac{1}{2}\|\boldsymbol{w} + \phi(\boldsymbol{y}, \boldsymbol{y}_i, \boldsymbol{x}_i) d\|^2 + \frac{1}{4C}\left(d + \sum_{\boldsymbol{y} \in \mathcal{A}_i} \alpha_{i,\boldsymbol{y}}\right)^2 - d\Delta(y_i, \boldsymbol{y}) \\ &\qquad \text{s.t.} \quad \alpha_{i,\boldsymbol{y}} + d \geq 0. \end{aligned} \tag{4.1}$$

Here, $\boldsymbol{w}$ is defined in Eq. (2.15) and $\boldsymbol{e}_{i,\boldsymbol{y}}$ is a vector where only the element corresponding to $(i, \boldsymbol{y})$ is one and the rest are zero. $D(\cdot)$ is defined in (2.14). Eq. (4.1) is a quadratic optimization problem with one variable and has an analytic solution. Therefore, the update rule of $\alpha_{i,\boldsymbol{y}}$ can be written as:

$$\begin{aligned} \bar{d}_{i,\boldsymbol{y}} &\leftarrow \frac{\Delta(\boldsymbol{y}, \boldsymbol{y}_i) - \boldsymbol{w}^T \phi(\boldsymbol{y}, \boldsymbol{y}_i, \boldsymbol{x}_i) - \frac{1}{(2C)} \sum_{\boldsymbol{y}'} \alpha_{i,\boldsymbol{y}'}}{\|\phi(\boldsymbol{y}, \boldsymbol{y}_i, \boldsymbol{x}_i)\|^2 + \frac{1}{(2C)}}, \\ \alpha_{i,\boldsymbol{y}} &\leftarrow \max(\alpha_{i,\boldsymbol{y}} + \bar{d}_{i,\boldsymbol{y}}, 0). \end{aligned} \tag{4.2}$$

To maintain the relation between $\boldsymbol{\alpha}$ and $\boldsymbol{w}$ specified in Eq. (2.15), $\boldsymbol{w}$ is updated accordingly:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + d_{i,\boldsymbol{y}} \phi(\boldsymbol{y}, \boldsymbol{y}_i, \boldsymbol{x}_i). \tag{4.3}$$

We will now discuss two implementation issues to improve DEMI-DCD. First, during the learning, $\boldsymbol{w}$ is shared between the learning and the active set selection threads. Therefore, we would like to maintain $\boldsymbol{w}$ in a shared buffer. However, the update rules in Steps (4.2)-(4.3) can be done in $O(\bar{n})$, where $\bar{n}$ is number of average active features of $\phi(\boldsymbol{x}, \boldsymbol{y})$. Thus, when the number of active

Algorithm 3 Learning Thread

Require: Dataset $\mathcal{D}$ and the number of iterations before updating the shared buffer, ρ.
Ensure: The learned model $\boldsymbol{w}$

```
w ← 0, α ← 0, #updates ← 0.
while stopping conditions are not satisfied do
  for i = 1 → l (loop over each instance) do
    for all y in A_i do
      if Eq. (4.4) is satisfied then
        A_i ← A_i \ {α_{i,y}}.
      else
        update corresponding α_{i,y} by Eq. (4.2)-Eq.(4.3),
        #updates ← #updates + 1.
      end if
      if #updates mod ρ = 0 then
        Copy w to w̄ in a shared buffer.
      end if
    end for
  end for
end while
```

features is small, the updates are quick. Hence, we require to maintain a lock to prevent the active set selection threads from accessing $\boldsymbol{w}$ when it is being updated. To reduce the cost of synchronization, we maintain a local copy of $\boldsymbol{w}$ and copy $\boldsymbol{w}$ to a shared buffer (denoted by $\bar{\mathsf{w}}$) after every ρ updates. This reduces the overhead of synchronization.

Second, as the active set selection thread keeps adding dual variables into $\mathcal{A}$, the size of $\mathcal{A}$ grows quickly. To avoid the learning thread from wasting time on the bounded dual variables, we implement a shrinking strategy inspired by (Joachims, 1999)[1]. Specifically, if $\alpha_{i,\bar{\boldsymbol{y}}}$ equals to zero and

$$-\nabla(\boldsymbol{\alpha})_{i,\bar{\boldsymbol{y}}} = \Delta(\bar{\boldsymbol{y}}, \boldsymbol{y}_i) - \boldsymbol{w}^T \phi(\bar{\boldsymbol{y}}, \boldsymbol{y}_i, \boldsymbol{x}_i) - \frac{1}{2} \sum_{\boldsymbol{y} \in \mathcal{A}_i} \alpha_{i,\boldsymbol{y}} < \delta \tag{4.4}$$

then DEMI-DCD removes $\alpha_{i,\bar{\boldsymbol{y}}}$ from $\mathcal{A}_i$. Notice that the shrinking strategy is more aggressive if δ is large. For binary classification, a negative δ is usually used. This is because, in the binary classification case, the size of the data is usually large (typically millions of examples); therefore incorrectly removing an instance from the active set requires a lengthy process that iterates over all the examples to add it back. However, in our case, aggressive shrinking strategy is safe because the active set selection threads can easily add the candidate structures back. Therefore, in our

[1]This shrinking strategy is also related to the condition used by cutting plane methods for adding constraints into the working set.

Algorithm 4 The j^{th} Active Set Selection Thread

Require: Part of dataset for this thread $\mathcal{B}_j$, and the number of iterations before updating the shared buffer, ρ.

1: $\boldsymbol{w} \leftarrow \mathbf{0}$ (a local copy), #Inference $\leftarrow 0$,
2: **while** Learning thread is not stopped **do**
3: **for all** $(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{B}_j$ **do**
4: $\bar{\boldsymbol{y}} \leftarrow f_{\text{AugInf}}(\boldsymbol{w}, \boldsymbol{x}_i, \boldsymbol{y}_i)$.
5: **if** $\bar{\boldsymbol{y}} \notin \mathcal{A}_i$ **then**
6: $\mathcal{A}_i \leftarrow \{\mathcal{A}_i \cup \bar{\boldsymbol{y}}\}$.
7: **end if**
8: #Inference $\leftarrow$ #Inference + 1
9: **if** #Inference mod $\rho = 0$ **then**
10: Copy from the model $\bar{\mathbf{w}}$ in shared buffer to $\boldsymbol{w}$.
11: **end if**
12: **end for**
13: **end while**

implementation, we set δ to 0.01.

Active Set Selection Threads. As mentioned, the j^{th} active set selection thread iterates over all instances $\boldsymbol{x}_i \in B_j$ and selects candidate active variables based on solving an augmented inference problem (line 4 in Algorithm 4), that is, eq. (2.16). Just like the learning thread, each active set selection thread maintains a local copy of $\boldsymbol{w}$. This setting aims to prevent $\boldsymbol{w}$ from being changed while the loss augmented inference is being solved, thus avoiding a possibly suboptimal solution. The local copy of $\boldsymbol{w}$ will be updated from $\bar{\mathbf{w}}$ in the shared buffer after each ρ iterations.

Synchronization. Our algorithm requires little synchronization between threads. In fact, only the learning thread can write to $\bar{\mathbf{w}}$ and only the j^{th} active set selection thread can modify $\mathcal{A}_i$ for any $i \in B_j$. It is very unlikely, but possible, that the learning thread and the inference threads will be reading/writing $\bar{\mathbf{w}}$ or $\mathcal{A}_i$ concurrently. To avoid this, one can use a mutex lock to ensure that the copy operation is atomic. However, in practice, we found that this synchronization is unnecessary.

4.1.1 Analysis

In this section, we analyze the proposed multi-core algorithm. We observed that α's can be added to the $\mathcal{A}$ by the active set selection thread, but might be removed by the learning thread with the shrinking strategy. Therefore, we define $\bar{\mathcal{A}}$ to be a subset of $\mathcal{A}$ that contains $\alpha_{i,\boldsymbol{y}}$ which has been visited by the learning thread at least once and remains in the $\mathcal{A}$.

Theorem 3. *The number of variables which have been added to $\bar{\mathcal{A}}$ during the entire training process is bounded by $O(1/\delta^2)$.*

The proof follows from (Chang and Yih, 2013, Theorem 1). This theorem says that the size of $\bar{\mathcal{A}}$ is bounded as a function of δ.

Theorem 4. *If $\bar{\mathcal{A}} \neq \emptyset$ is not expanded, then the proposed algorithm converges to an ϵ-optimal solution of*

$$\min_{\boldsymbol{\alpha} \geq 0} \quad D(\boldsymbol{\alpha}) \quad s.t.\ \alpha_{i,\boldsymbol{y}} = 0, \forall \boldsymbol{y} \notin \mathcal{A}_i \tag{4.5}$$

in $O(log(1/\epsilon))$ steps.

If $\bar{\mathcal{A}}$ is fixed, our learning thread performs standard dual coordinate descent, as in (Hsieh et al., 2008). Hence, this theorem follows from the analysis of dual coordinate descent. The global convergence rate of the method can be inferred from (Wang and Lin, 2013) which generalizes the proof in (Luo and Tseng, 1993).

Theorem 3 shows that the size of $\bar{\mathcal{A}}$ will eventually stop growing. Theorem 4 shows that when $\bar{\mathcal{A}}$ is fixed, the weight vector $\boldsymbol{w}$ converges to the optimum of (4.5). Hence, the local copies in the learning thread and the active set selection threads can be arbitrary close. Following the analysis in (Chang and Yih, 2013), the convergence of DEMI-DCD then follows.

4.2 Related Work

Several related works have a resemblance to the method proposed in this chapter. In the following, we briefly review the literature and discuss the connections.

A Parallel Algorithm for Dual Coordinate Descent. The structured SVM package JLIS (Chang et al., 2010) implements a parallel algorithm in a Master-Slave architecture to solve Eq. (2.14).[2], Given p processors, it first splits the training data into p parts. Then the algorithm maintains a model $\boldsymbol{w}$, dual variables $\boldsymbol{\alpha}$, and an active set $\mathcal{A}$ and updates these in an iterative fashion. In each iteration, the master thread sends one part of the data to a slave thread. For each slave thread, it iterates over each assigned example $\boldsymbol{x}_i$, and picks the best structures $\bar{\boldsymbol{y}}$ according

[2]The implementation of MS-DCD can be downloaded at `http://cogcomp.cs.illinois.edu/page/software_view/JLIS`.

to current $\boldsymbol{w}$. Then $(\boldsymbol{x}_i, \bar{\boldsymbol{y}})$ is added into the active set $\mathcal{A}$ if the following condition is satisfied:

$$\Delta(\bar{\boldsymbol{y}}, \boldsymbol{y}_i) - \boldsymbol{w}^T \phi(\bar{\boldsymbol{y}}, \boldsymbol{y}_i, \boldsymbol{x}_i) - \frac{1}{2} \sum_{\boldsymbol{y} \in \mathcal{A}_i} \alpha_{i,\boldsymbol{y}} > \delta. \tag{4.6}$$

Only after all the slave threads have finished processing all the examples, the master thread performs dual coordinate descent updates to solve the following optimization loosely:

$$\min_{\boldsymbol{\alpha} \geq 0} \quad D(\boldsymbol{\alpha}) \quad \text{s.t. } \alpha_{i,\boldsymbol{y}} = 0, \forall \boldsymbol{y} \notin \mathcal{A}_i.$$

The algorithm stops when a stopping condition is reached. This approach is closely related to the n-slack cutting plane method for solving structured SVM (Tsochantaridis et al., 2005). However, Tsochantaridis et al. (2005) assumes that the sub-problem is solved exactly[3], while this restriction can be relaxed under a dual coordinate descent framework.

We will refer to this approach for parallelizing structured SVM training as MS-DCD (for Master-Slave dual coordinate descent) and compare it experimentally with DEMI-DCD in Section 4.3.

Parallel Structured Perceptron. A parallelization strategy for structured Pereceptron (SP-IPM) has been proposed (McDonald et al., 2010b) using the Map-Reduce framework. It calls for first splitting the training data into several parts. In the map phase, it distributes data to each mapper and runs a separate Perceptron learner on each shard in parallel. Then, in the reduce phase, the models are mixed using a linear combination. The mixed model serves as the initial model for the next round. In this thesis, we implement this map-reduce framework as a multi-thread program. SP-IPM requires barrier synchronizations between the map and reduce phases, which limits the computational performance. In addition, in the model mixing strategy, each local model is updated using exclusive data blocks. Therefore, the mappers might make updates that are inconsistent with each other. As a result, it requires many iterations to converge.

General Parallel Algorithms for Convex Optimization. Some general parallel optimization algorithms have been proposed in the literature. For example, delayed stochastic (sub-)gradient descent methods have been studied with assumptions on the smoothness of the problem (Agarwal

[3]The cutting-plane solver for structured SVM usually sets a tolerance parameter, and stops the sub-problem solver when the inner stopping criteria is satisfied.

and Duchi, 2011; Langford et al., 2009b). However, their applicability to structured SVM has not been explored.

4.3 Experiments

In this section, we show the effectiveness of the DEMI-DCD algorithm compared to other parallel structured learning algorithms.

4.3.1 Experimental Setup

We evaluate our algorithm on two natural language processing tasks: part-of-speech tagging (POS-WSJ) and jointly predicting entities and their relations (Entity-Relation). These tasks, which have very different output structures, are described below.

POS Tagging (POS-WSJ). POS tagging is the task of labeling each word in a sentence with its part of speech. This task is typically modeled as a sequence labeling task, where each tag is associated with emission features that capture word-tag association and transition features that capture sequential tag-tag association. For this setting, inference can be solved efficiently using the Viterbi algorithm.

We use the standard Penn Treebank Wall Street Journal corpus (Marcus et al.) to train and evaluate our POS tagger using the standard data split for training (sections 2-22, 39832 sentences) and testing (section 23, 2416 sentences). In our experiments, we use indicators for the conjunction between the word and tags as emission features and pairs of tags as transition features.

Entity and Relation Recognition (Entity-Relation). This is the task of assigning entity types to spans of text and identifying relations among them (Roth and Yih, 2004). For example, for the sentence *John Smith resides in London*, we would predict *John Smith* to be a PERSON, LONDON to be a LOCATION and the relation LIVES-IN between them.

As in the original work, we modeled prediction as a 0-1 linear program (ILP) where binary indicator variables capture all possible entity-label decisions and entity pair label decisions (that is, relation labels). Linear constraints force exactly one label to be assigned to each entity and relation. In addition, the constraints also encode background knowledge about the types of entities allowed for each relation label. For example, a LIVES-IN relation can only connect a PERSON to a

LOCATION. We modeled the prediction as a 0-1 integer linear program, where the binary variables represent the possible assignments. We use the annotated corpus from Roth and Yih (2007), which consists of 5,925 sentences. Unlike the original paper, which studied a decomposed learning setting, we jointly train the entity-relation model using an ILP for the loss-augmented inference. We used the state-of-the-art Gurobi ILP solver (Gurobi Optimization, Inc., 2012) to solve inference for this problem both during training and test. We report results using the annotated data from (Roth and Yih, 2007) consisting of 5925 examples with an 80-20 train-test split.

We based our implementation on the publicly available implementation of MS-DCD, which implements a dual coordinate descent method for solving structured SVM. DCD (Chang and Yih, 2013) has been shown competitive comparing to other L2-loss and L1-loss structured SVM solvers such as a 1-slack variable cutting-plane method (Joachims et al., 2009) and a Frank-Wolfe optimization method (Lacoste-Julien et al., 2013) when using one CPU core.

As described in Section 4.1, our method is an extension of DCD and further improves its performance using multiple cores. All the algorithms are implemented in Java. We conducted our experiments on a 24-core machine with Xeon E5-2440 processors running 64-bit Scientific Linux. Unless otherwise stated, all results use 16 threads. We set the value of C to 0.1 for all experiments.

Our experiments compare the following three methods:

1. DEMI-DCD: the proposed algorithm described in Section 4.1. We use one thread for learning and the rest for inference (that is, active set selection).

2. MS-DCD: A master-slave style parallel implementation of dual coordinate descent method from JLIS package (Chang et al., 2010) described in Section 4.2.

3. SP-IPM: A parallel structured Perceptron algorithm proposed in (McDonald et al., 2010b). We run 10 epochs of Perceptron updates for each shard at each outer iteration.

Note that the first two methods solve an L2-Structural SVM problem (2.13) and converge to the same minimum.

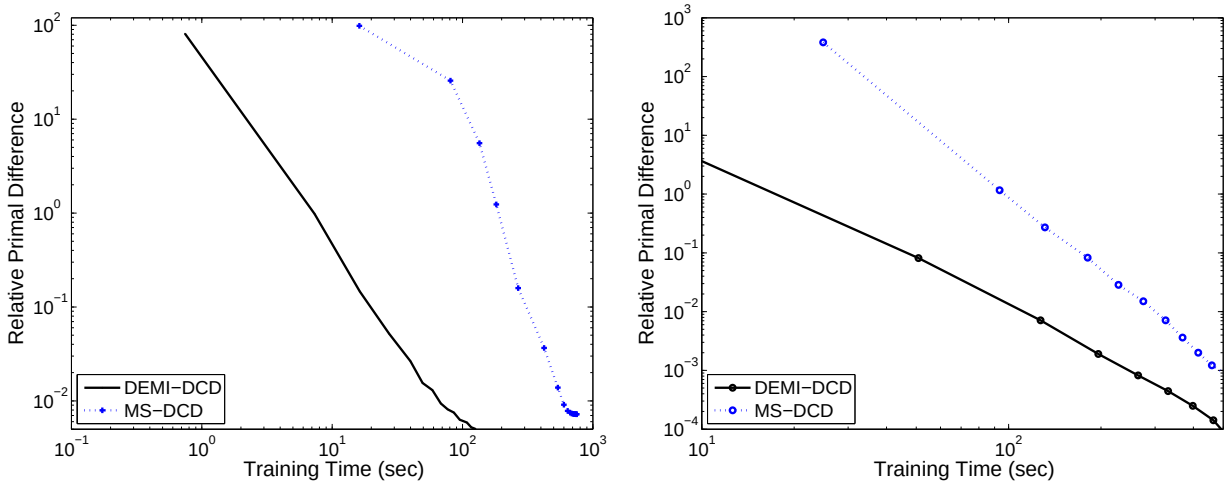
Our experiments answer the following research question: Can DEMI-DCD make use of the available CPU resources to converge faster to a robust structured prediction model? To answer this, for both our tasks, we first compare the convergence speed of DEMI-DCD and MS-DCD.

Second, we compare the performance of the three algorithms on the test sets of the two tasks. Third, we show that DEMI-DCD maximally utilizes all available CPU cores unlike the other two algorithms. Finally, we report the results of an analysis experiment, where we show the performance of our algorithm with different number of available threads.

4.3.2 Empirical Convergence Speed

First, we compare DEMI-DCD to MS-DCD in terms of their speed of convergence in terms of objective function value. We omit SP-IPM in this comparison because it does not solve the SVM optimization problem. Figure 4.1 shows the relative primal objective function (that is, the value of the objective in Eq. (2.13)) with respect to a reference model $\boldsymbol{w}^*$ as a function of wall-clock training time. In other words, if the objective function is denoted by f, we plot $\left(f(\boldsymbol{w}) - f(\boldsymbol{w}^*)\right)/f(\boldsymbol{w}^*)$ as $\boldsymbol{w}$ varies with training time for each algorithm. The reference model is the one that achieves the lowest primal value among the compared models. In the figure, both the training time and the relative difference are shown in log-scale. From the results, we see that the proposed method is faster than MS-DCD on both tasks. DEMI-DCD is especially faster than MS-DCD in the early stage of optimization. This is important because usually we can achieve a model with reasonable generative performance before solving (2.13) exactly. Note that the inference in Entity-Relation is much slower than inference for POS-WSJ. For example, at each iteration on POS-WSJ, MS-DCD takes 0.62 seconds to solve inference on all the training samples using 16 threads and takes 3.87 seconds to update the model using 1 thread, while it takes 10.64 seconds to solve the inference and 8.45 seconds to update the model on Entity-Relation. As a result, the difference between DEMI-DCD and MS-DCD is much higher on POS-WSJ. We will discuss the reason for this in Section 4.3.3.

Next, we show that the proposed method can obtain a reasonable and stable solution in a short time. Figure 4.2 shows the test set performance of the three algorithms as training proceeds. For POS-WSJ, we evaluate the model using token-based accuracy. For Entity-Relation, we evaluate the entity and relation labels separately and report the micro-averaged F1 over the test set. In all cases, DEMI-DCD is the fastest one to achieve a converged performance. As mentioned, DEMI-DCD is more efficient than MS-DCD in the early iterations. As a result, it takes less time to generate



(a) POS-WSJ (b) Entity-Relation

Figure 4.1: Relative primal function value difference to the reference model versus wall clock time. See the text for more details.

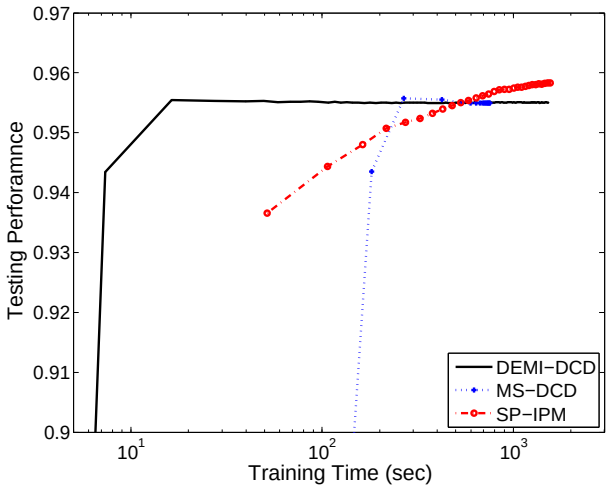
a reasonable model. Note that SP-IPM achieves better final performance on the POS-WSJ task. This is so because SP-IPM converges to a different model from DEMI-DCD and MS-DCD. For both entities and relations, SP-IPM converges slowly and, moreover, its performance is unstable for the relations. We observe that the convergence of SP-IPM is slow because the Perceptron algorithm needs to solve an inference problem before each update. When the inference is slow, the number of updates in SP-IPM is significantly smaller than DEMI-DCD and MS-DCD. For example, in the Entity-Relation task, DEMI-DCD makes around 55 million updates within 100 seconds, while SP-IPM only makes 76 thousand updates in total.[4] In other words, DEMI-DCD is faster and better than SP-IPM because it performs many inexpensive updates.

4.3.3 CPU Utilization

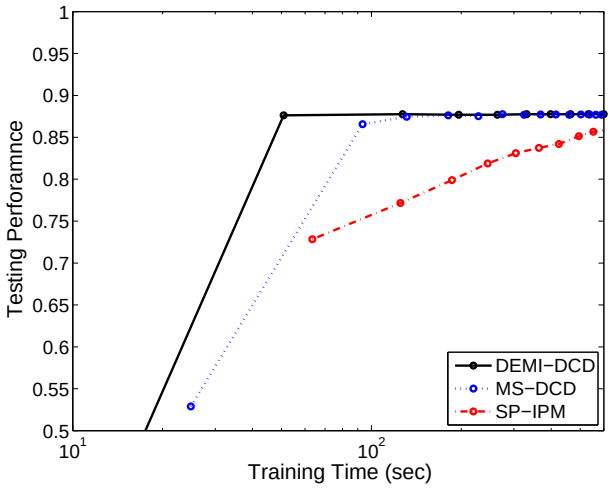
Finally, we investigate the CPU utilization of the three algorithms by plotting the *moving average* of CPU usage in Figure 4.3 during training, as reported by the Unix command `top`. Since we provide all algorithms with 16 threads, the maximum CPU utilization can be 1600%. The results show that DEMI-DCD almost fully utilizes the available CPU resources.

We see that neither baseline manages to use the available resources consistently. The average CPU usage for MS-DCD on POS-WSJ is particularly small because the inference step on this task

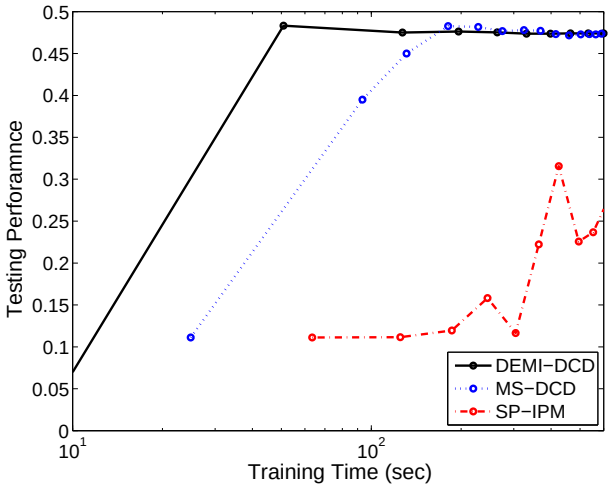
[4]SP-IPM has 16 threads running on different shards of data in parallel. We simply sum up all the updates on different shards.



(a) POS-WSJ: Token-wise accuracy



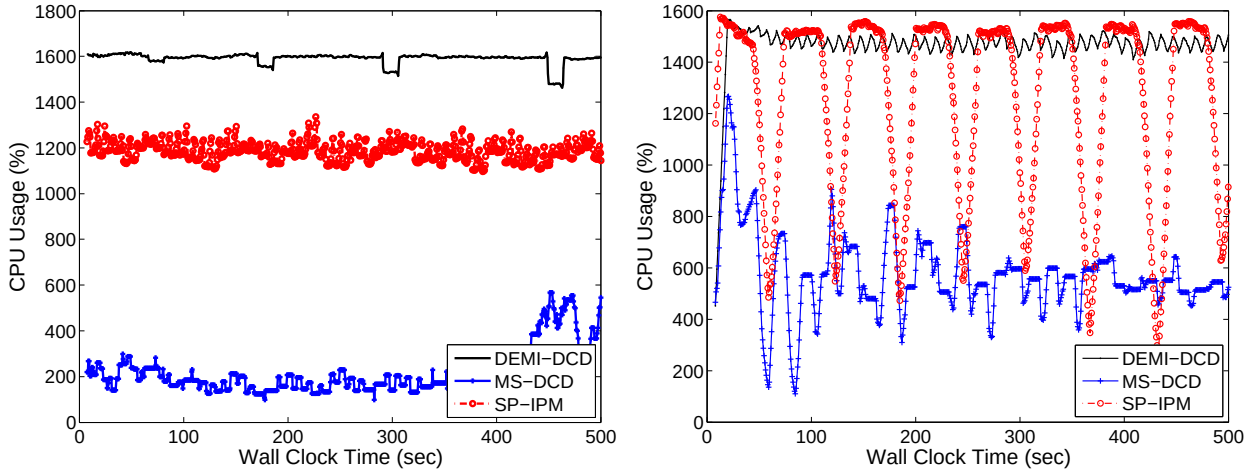
(b) Entity-Relation: Entity F1



(c) Entity-Relation: Relation F1

Figure 4.2: Performance of POS-WSJ and Entity-Relation plotted as a function of wall clock training time. For Entity-Relation, we report the F1 scores of both entity and relation tasks. Note that the X-axis is in **log scale**. We see that DEMI-DCD converges faster to better performing models.

is relatively easy (i.e. using the Viterbi algorithm). In fact, MS-DCD spends only around 25% of the time on the inference steps, which is solved by the slave threads in parallel. Both MS-DCD and SP-IPM require a barrier to ensure that the subtasks sent to the slave threads are complete. This barrier limits the CPU utilization and hence slows down the learners. Since inference is more expensive (i.e. an ILP call) for the Entity-Relation case, more time is spent on inference in the learning algorithms. Since this step is distributed across the slaves for both MS-DCD and SP-



(a) POS-WSJ (1,318%, 257%, 1,177%) (b) Entity-Relation (1,462%, 527%, 1248%)

Figure 4.3: CPU usage of each method during training. The numbers listed in the caption are the average CPU usage percentage per second for DEMI-DCD, MS-DCD, SP-IPM, respectively. We show moving averages of CPU usage with a window of 2 seconds. Note that we allow all three algorithms to use 16 threads in a 24 core machine. Thus, while all the algorithms can report upto 1600% CPU usage, only DEMI-DCD consistently reports high CPU usage.

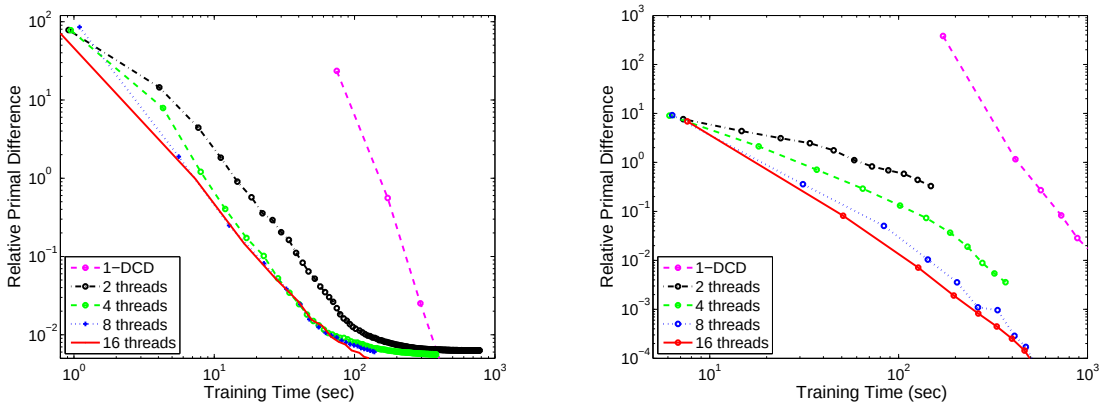
IPM, we see periods of high CPU activity followed by low activity (when only the master thread is active).

4.3.4 Performance with Different Number of Threads

In our final set of experiments, we study the performance of DEMI-DCD for different number of threads. Figure 4.4 shows the change in the primal objective function value difference as a function of training time for different number of threads. Note that the training time and the function values are shown in *log-scale*. For comparison, we also show the performance of a DCD implementation using only one CPU core (1-DCD). As can be seen in the figure, the training time is reduced as the number of threads increases. With multiple threads, DEMI-DCD is significantly faster than 1-DCD. However, when the number of threads is more than 8, the difference is small. That is because the inference step (i.e, the active set selection step) is no longer the bottleneck.

4.4 Conclusion

In this chapter, we have proposed a new learning algorithm for training structured SVMs, called DEMI-DCD. This algorithm decouples the model update and inference phases of learning allowing us



(a) POS-WSJ (b) Entity-Relation

Figure 4.4: Relative primal function value difference along training time using different number of threads. We also show a DCD implementation using one CPU core (1-DCD). Both x-axis and y-axis are in **log scale**.

to execute them in parallel, thus enabling us to take advantage of multi-core machines for training. We showed that the DEMI-DCD algorithm converges to the optimum solution of the structured SVM objective. We experimentally evaluated our algorithm on the structured learning tasks of part-of-speech tagging and entity-relation extraction and showed that it outperforms existing strategies for training structured predictors in terms of both convergence time and CPU utilization.

Chapter 5

Amortized Inference for Structured Prediction

In the Chapter 4, we showed a parallel algorithm for training structured SVM models. In this chapter, we describe a general strategy to speed up structured learning algorithms by caching inference solutions. Training a structured prediction model involves performing several (loss-augmented) inference steps, where the algorithm finds the best structure according to the current model for a training example. Existing learning algorithms (Tsochantaridis et al., 2005; Joachims et al., 2009; Shevade et al., 2011; Chang and Yih, 2013; Lacoste-Julien et al., 2013) often treat the inference procedure as a black-box and solve the inference problems encountered during training *independently*. However, a learning algorithm makes multiple passes over the training examples, and updates its parameters for each example several times. Consequently, the inference problems it encounters may often have identical or similar solutions. To verify this intuition, we trained a structured SVM model on a joint entity-relation recognition task (Roth and Yih, 2007) and plotted the numbers of inference engine calls (black solid line) made in Figure 5.1. As shown in the figure, a large fraction of these calls returns identical solutions, as indicated by the blue dotted line, even when the objective functions of the inference problems are different. This suggests the possibility of amortizing these inference steps (see details in caption of Figure 5.1).

In Chang et al. (2015), we apply the idea of selection and caching for exploring the redundancy of inference solutions involved in the learning. As a result, we are able to *amortize the inferences during learning*, accelerating the training process of a structured model. That is, we measure the overall cost of inference over the lifetime of the algorithm, and show that this "amortized" cost can be significantly reduced. Our framework only depends on the fact that the inference is formulated as an Integer Linear Program (ILP), a very general condition, as discussed below, that does not restrict its applicability. Given an ILP formulation of an inference problem, there are some conditions that determine if this inference problem shares the same solution with previously observed

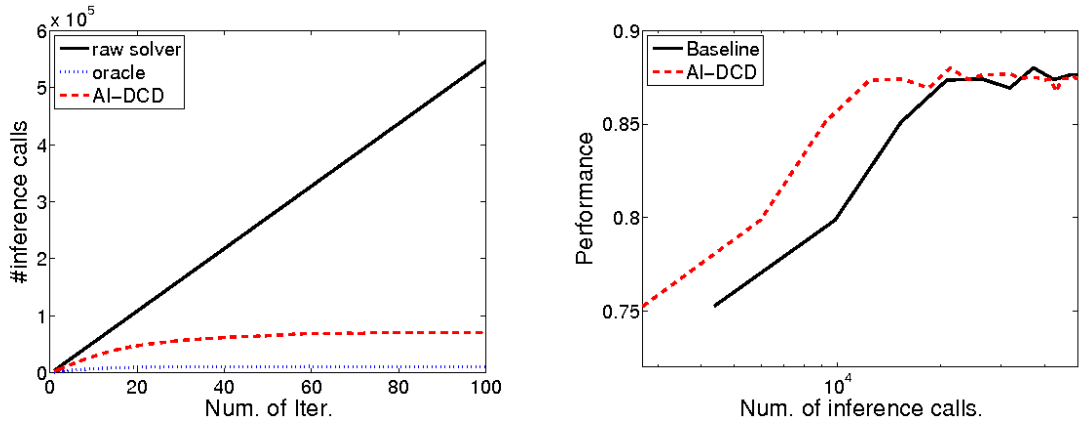


Figure 5.1: Left: The number of inference engine calls by a raw solver, an oracle solver and our method when training a structured SVM model on an entity-relation recognition corpus. The raw solver calls an inference engine each time an inference is needed, while the oracle solver only calls the engine when the solution is different from all solutions observed previously. Our method significantly reduces the number of inference calls required to train a model. Right: Our models require less inference calls to reach a certain level of performance.

problems (Srikumar et al., 2012; Kundu et al., 2013; Chang et al., 2015). This property allows us to reuse solutions of previously solved problems by caching the coefficients of their objective functions and their solutions.

We first introduce amortized inference theorems that guarantee that two *different* inference problems posed as ILPs will yield the same or similar solutions if the conditions are satisfied. Based on the bound of inference quality, we then show that when amortized inference is plugged in a dual coordinate descent solver (DCD) for structured SVM, the empirical risk bound is guaranteed. We provide sound theoretical properties of the proposed methods and show empirically (Figure 5.1), that our framework can significantly reduce the number of inference calls required for reaching a certain level of performance (red dashed lines) when training the same structured SVM model without any drop in performance (black solid line).

Our learning framework is related to approximate training paradigms (Finley and Joachims, 2008; Meshi et al., 2010; Hazan and Urtasun, 2010; Sutton and McCallum, 2007; Samdani and Roth, 2012), but with some crucial differences. These methods use approximate inference to find suboptimal solutions to otherwise intractable inference problems. In contrast, we focus on scenarios where exact solutions can be computed, developing an approximate scheme to accelerate training without affecting the solution quality. Caching inference solutions has been discussed in a different

context for 1-slack cutting plane method (Joachims et al., 2009), where cached solutions can be used to generate a new cutting-plane. We will discuss this in later sections.

5.1 Amortized Inference

As we mentioned in Chapter 2, we let $\boldsymbol{y} = \{y_1, y_2, \cdots y_N\}$ be a structured output object, where $y_j \in \mathcal{Y}_j$, a discrete set of candidates for the output variable y_j. And we use a binary vector $\boldsymbol{z} = \{z_{j,a} \mid j = 1 \cdots N, a \in \mathcal{Y}_j\}, \boldsymbol{z} \in \mathcal{Z} \subset \{0,1\}^{\sum |\mathcal{Y}_i|}$ to represent the output variable $\boldsymbol{y}$, where $z_{j,a}$ denotes whether y_j assumes the value a or not. This way (2.10) can be reformulated as a binary integer linear program (ILP) (Eq. 2.12):

$$\arg\max_{\boldsymbol{z} \in \mathcal{Z}} \quad \boldsymbol{c}^T \boldsymbol{z},$$

where $\boldsymbol{c}^T \boldsymbol{z} = \boldsymbol{w}^T \phi(\boldsymbol{x}, \boldsymbol{y})$. In this section, we assume that the coefficient vector $\boldsymbol{c}$ of the objective function is given. The values of $\boldsymbol{c}$ are determined by $\boldsymbol{w}$ and $\boldsymbol{x}$. $\mathcal{Z}$ represents the feasible set of the inference problem. Given an inference problem P, $[P]$ represents an *equivalence class* of integer linear programs which have the same number of variables and the same feasible set. In general, the constraints in 2.12 do not depend on the model $\boldsymbol{w}$ and the input $\boldsymbol{x}$. Therefore, inference problems with the same output size are in the same equivalent class. For example, in a part-of-speech tagging task, inference problems corresponding to sentences with the same number of words are in the same equivalent class.

Srikumar et al. (2012) observed that (1) different objective functions often yield the same solution, and (2) only a small fraction of the exponentially many possible structures is actually seen in practice. Based on these observations, they derive *amortized inference theorems* that characterize the set of ILP objectives which share the same solution. In the following, we suppose that we have two inference problems $\mathbf{p}$ and $\mathbf{q}$, with corresponding objective coefficients $\boldsymbol{c}^{\mathbf{p}}$ and $\boldsymbol{c}^{\mathbf{q}}$, respectively. We have solved the problem $\mathbf{p}$ and have obtained its solution $\boldsymbol{y}^{\mathbf{p}}$, with corresponding binary representation $\boldsymbol{z}^{\mathbf{p}}$. We further assume that the problems $\mathbf{p}$ and $\mathbf{q}$ are in the same equivalent class. In the following section, we discuss the exact amortized inference theorems proposed in (Srikumar et al., 2012), which can be used to determine if problems $\mathbf{p}$ and $\mathbf{q}$ share the same

solution. Then, we discuss how to derive approximate amortized inference theorems by relaxing the conditions (Chang et al., 2015).

5.1.1 Exact Amortized Inference Theorems

A Theorem based on the Sensitivity to the Perturbations of Objective Coefficients. The first amortized inference theorem is based on the following intuition. Given an integer linear programming problem $\mathbf{p}$ and its solution $\boldsymbol{z}^{\mathbf{p}}$. For every i, a in $\{(i,a) \mid z^{\mathbf{p}}_{i,a} = 1\}$, increasing its associated objective coefficient $c_{i,a}$ will not change the optimal assignment of $z^{\mathbf{p}}_{i,a}$. Similarly, if $z^{\mathbf{p}}_{i,a} = 0$, decreasing $c_{i,a}$ will not change the optimal assignment of $z^{\mathbf{p}}_{i,a}$. Based on this intuition, the following amortized inference theorem (Theorem 1 in (Srikumar et al., 2012)) holds.

Theorem 5. *Let* $\mathbf{p}$ *and* $\mathbf{q}$ *be inference problems that share the same feasible set and have the same number of variables (i.e.,* $\mathbf{q} \in [\mathbf{p}]$*). Let* $\boldsymbol{y}^{\mathbf{p}} = \{y^{\mathbf{p}}_1, y^{\mathbf{p}}_2, ...y^{\mathbf{p}}_N\}$[1] *and* $\boldsymbol{y}^{\mathbf{q}} = \{y^{\mathbf{q}}_1, y^{\mathbf{q}}_2, ...y^{\mathbf{q}}_N\}$ *be the optimal solutions to the inference problems* $\mathbf{p}$ *and* $\mathbf{q}$ *respectively, with corresponding binary representations* $\boldsymbol{z}^{\mathbf{p}}$ *and* $\boldsymbol{z}^{\mathbf{q}}$*. Denote the objective function of* $\mathbf{p}$ *by* $f^{\mathbf{p}}(\boldsymbol{z}) = \boldsymbol{c}^{\mathbf{p}} \cdot \boldsymbol{z}$*, and of* $\mathbf{q}$ *by* $f^{\mathbf{q}}(\boldsymbol{z}) = \boldsymbol{c}^{\mathbf{q}} \cdot \boldsymbol{z}$*. If* $\boldsymbol{c}^{\mathbf{p}}$ *and* $\boldsymbol{c}^{\mathbf{q}}$ *satisfy the following condition:*

$$(2z^{\mathbf{p}}_{j,a} - 1)(c^{\mathbf{p}}_{j,a} - c^{\mathbf{q}}_{j,a}) \leq 0, \quad \forall j, a \tag{5.1}$$

then $\boldsymbol{z}^{\mathbf{p}}$ *is an optimal solution to the inference problem* $\mathbf{q}$*.*

Note that

$$2z^{\mathbf{p}}_{j,a} - 1 = \begin{cases} 1 & \text{if } z^{\mathbf{p}}_{j,a} = 1, \\ -1 & \text{if } z^{\mathbf{p}}_{j,a} = 0. \end{cases}$$

This theorem shows that if the changes in the coefficients of the objective function satisfy Condition (5.1), then the optimal solution will not change.

A Theorem based on the Convex Cone of the Coefficients. Srikumar et al. (2012) derive another amortize inference algorithm by the following intuition. Let a set of ILP problems $\{\mathbf{p}^j\}$

[1] The value of the j-th element in the solution $\boldsymbol{y}^{\mathbf{p}}$ is $y^{\mathbf{p}}_j$ (i.e., $z_{j,y^{\mathbf{p}}_j} = 1$).

shares the same solution $\boldsymbol{z}^{\mathbf{p}}$. For any j, we have $\boldsymbol{c}^{\mathbf{p}^j} \cdot \boldsymbol{z}^{\mathbf{p}} \geq \boldsymbol{c}^{\mathbf{p}^j} \cdot \boldsymbol{z}', \forall \boldsymbol{z}' \in \mathcal{Z}$. Then, for any $\{x_j \geq 0\}$

$$\sum_j x_j \boldsymbol{c}^{\mathbf{p}^j} \cdot \boldsymbol{z}^{\mathbf{p}} \geq \sum_j x_j \boldsymbol{c}^{\mathbf{p}^j} \cdot \boldsymbol{z}', \forall \boldsymbol{z}' \in \mathcal{Z}.$$

This indicates $\boldsymbol{z}^{\mathbf{p}}$ is the optimal solution to any inference problems $\mathbf{q} \in [\mathbf{p}]$ with coefficients $\boldsymbol{c}^{\mathbf{q}}$ that lie in the convex cone formed by $\{\mathbf{p}^j\}$ (i.e., $\boldsymbol{c}^{\mathbf{q}} = \sum_j x_j \boldsymbol{c}^{\mathbf{p}^j}$.) We formally write down the theorem below.

Theorem 6. *Let a set of inference problems* $\{\mathbf{p}^j\}$ *with corresponding coefficients* $\{\boldsymbol{c}^{\mathbf{p}^j}\}$ *share the same solution* $\boldsymbol{z}^{\mathbf{p}}$. $\boldsymbol{z}^{\mathbf{p}}$ *is the optimal solution to any inference problem* $\mathbf{q} \in [\mathbf{p}]$ *with coefficients*

$$\boldsymbol{c}^{\mathbf{q}} = \sum_j x_j \boldsymbol{c}^{\mathbf{p}^j}, \exists x_j \geq 0.$$

Theorems 5 and 6 can be combined. We refer the interested readers to (Srikumar et al., 2012) for details.

Margined-Based Amortized Inference Framework. Kundu et al. (2013) describe an amortized inference framework based on examination of the structured margin of a prediction. Given an inference problem $\mathbf{p}$ and its solution $\boldsymbol{z}^{\mathbf{p}}$, the structured margin δ of an inference problem $\mathbf{p}$ measures the difference in objective values between the best and the second best solutions. Specifically, the structured margin is defined as follows:

$$\delta = \min_{\boldsymbol{z} \in \mathcal{Z}, \boldsymbol{z} \neq \boldsymbol{z}^{\mathbf{p}}} \boldsymbol{c}^{\mathbf{p}} \cdot (\boldsymbol{z}^{\mathbf{p}} - \boldsymbol{z}). \tag{5.2}$$

This ensures that

$$\boldsymbol{c}^{\mathbf{p}} \cdot \boldsymbol{z}^{\mathbf{p}} - \boldsymbol{c}^{\mathbf{p}} \cdot \boldsymbol{z} \geq \delta, \ \forall \boldsymbol{z}.$$

Given a new inference problem $\mathbf{q} \in [\mathbf{p}]$, we further define Δ as the upper bound of the difference in objective values between problems $\mathbf{p}$ and $\mathbf{q}$ for which the solution is not $\boldsymbol{z}^{\mathbf{p}}$:

$$\Delta = \max_{\boldsymbol{z} \in \mathcal{Z}, \boldsymbol{z} \neq \boldsymbol{z}^{\mathbf{p}}} (\boldsymbol{c}^{\mathbf{q}} - \boldsymbol{c}^{\mathbf{p}}) \cdot \boldsymbol{z}. \tag{5.3}$$

Let $\Delta^{\mathbf{p}} = -(\boldsymbol{c}^{\mathbf{q}} - \boldsymbol{c}^{\mathbf{p}}) \cdot \boldsymbol{z}^{\mathbf{p}}$ be the decrease in the objective value for the solution $\boldsymbol{z}^{\mathbf{p}}$ when moving

from $\boldsymbol{c}^{\mathbf{p}}$ to $\boldsymbol{c}^{\mathbf{q}}$. If the sum of $\Delta^{\mathbf{p}}$ (the maximal decrease of objective value for $\boldsymbol{z}^{\mathbf{p}}$) and Δ (the maximal increase for other solutions) is less than δ (the difference between the objective values of the best and the second best solutions of $\mathbf{p}$), then the optimal solution of $\mathbf{q}$ is $\boldsymbol{z}^{\mathbf{p}}$. The following theorem captures this intuition:

Theorem 7. *Let* $\mathbf{p}$ *and* $\mathbf{q}$ *be inference problems belonging the same equivalent class* $[\mathbf{p}]$*. Let* $\boldsymbol{z}^{\mathbf{p}}$ *is the optimal solution of* $\mathbf{q}$*. Given* δ *and* Δ *defined in Eq.* (5.2) *and* (5.3)*, respectively. If the following condition holds:*

$$-(\boldsymbol{c}^{\mathbf{q}} - \boldsymbol{c}^{\mathbf{p}}) + \Delta \leq \delta \tag{5.4}$$

then $\boldsymbol{z}^{\mathbf{p}}$ *is the optimal solution of* $\mathbf{q}$*.*

Checking the validity of condition (5.4) requires solving another inference problem to compute Δ; However, this can be avoided by computing a relaxed version of (5.4) (see Kundu et al. (2013) for details). Note that the condition (5.4) is a strict generalization of condition (5.6).

5.1.2 Approximate Amortized Inference Theorems

The exact amortized inference theorems guarantee to obtain an optimal solution. However, in order to obtain a higher hit rate, we can consider relaxing the conditions and deriving approximate theorems. In the following, we study an approximate version of Theorem 5, which is shown to be useful when plugged it into the learning process Chang et al. (2015).

Theorem 8. *Let* $\mathbf{p}$ *and* $\mathbf{q}$ *be inference problems that share the same feasible set and have the same number of variables. Let* $\boldsymbol{y}^{\mathbf{p}} = \{y_1^{\mathbf{p}}, y_2^{\mathbf{p}}, ...y_N^{\mathbf{p}}\}$[2] *and* $\boldsymbol{y}^{\mathbf{q}} = \{y_1^{\mathbf{q}}, y_2^{\mathbf{q}}, ...y_N^{\mathbf{q}}\}$ *be the optimal solutions to the inference problems* $\mathbf{p}$ *and* $\mathbf{q}$ *respectively, with corresponding binary representations* $\boldsymbol{z}^{\mathbf{p}}$ *and* $\boldsymbol{z}^{\mathbf{q}}$*. Denote the objective function of* $\mathbf{p}$ *by* $f^{\mathbf{p}}(\boldsymbol{z}) = \boldsymbol{c}^{\mathbf{p}} \cdot \boldsymbol{z}$*, and of* $\mathbf{q}$ *by* $f^{\mathbf{q}}(\boldsymbol{z}) = \boldsymbol{c}^{\mathbf{q}} \cdot \boldsymbol{z}$*. W.l.o.g, assume* $f^{\mathbf{q}}(\boldsymbol{z}^{\mathbf{p}}) \geq 0$ *and* $f^{\mathbf{q}}(\boldsymbol{z}^{\mathbf{q}}) \geq 0$*. If there exists a constant* $M \in \mathcal{R}^+$ *such that*

$$M \geq \sum_j (|c^{\mathbf{q}}_{j,y_j^{\mathbf{p}}}| + |c^{\mathbf{q}}_{j,y_j^{\mathbf{q}}}|)/f^{\mathbf{q}}(\boldsymbol{z}^{\mathbf{p}}) \tag{5.5}$$

[2] The value of the j-th element in the solution $\boldsymbol{y}^{\mathbf{p}}$ is $y_j^{\mathbf{p}}$ (i.e., $z_{j,y_j^{\mathbf{p}}} = 1$).

and $\boldsymbol{c}^{\mathbf{p}}$ *and* $\boldsymbol{c}^{\mathbf{q}}$ *satisfy the following condition:*

$$(2z^{\mathbf{p}}_{j,a} - 1)(c^{\mathbf{p}}_{j,a} - c^{\mathbf{q}}_{j,a}) \leq \epsilon |c^{\mathbf{q}}_{j,a}|, \quad \forall j, a \tag{5.6}$$

then $\boldsymbol{z}^{\mathbf{p}}$ *is an* $(1/(1 + M\epsilon))$*-approximation to the inference problem* $\mathbf{q}$*. That is,* $f^{\mathbf{q}}(\boldsymbol{z}^{\mathbf{q}}) \leq (1 + M\epsilon) f^{\mathbf{q}}(\boldsymbol{z}^{\mathbf{p}})$*.*

The proof is provided in the appendix. To use this theorem, one would store a large cache of $(\mathbf{p}, \boldsymbol{c}^{\mathbf{p}})$ pairs with their solutions. Upon encountering a new inference problem $\mathbf{q}$, we look in the cache for $\mathbf{p}$ that satisfies (5.6). We thus obtain a $(1/(1 + M\epsilon))$-approximate solution to $\mathbf{q}$, saving a call to the inference engine. We later show that such approximate solutions suffice for training purposes. Note that this scheme is oblivious to the inference engine used. When the tolerance parameter $\epsilon = 0$, the solution obtained from the above process is exact. Otherwise, a large ϵ increases the number of inference problems that can be amortized, but also amplify the risk of obtaining suboptimal solutions.[3] The theorem assumes the optimal objective function value of $\mathbf{q}$ is positive. If the assumption is not satisfied, one can add a dummy variable with a large coefficient, and the theorem holds. Note that constructing an ILP problem for (2.12) and checking condition (5.6) are usually cheaper than solving Eq. (2.10).

We can derive an approximate version of Theorem 7. However, checking validity of (5.4) involves computing the structured margin (Δ) for each cached solution, which involves solving an additional ILP problem. If the solution cache is prepared in a batch setting (e.g., using amortized inference technique in the test phase), this one-time cost can be neglected. However, when we cache the solutions of inferences involved in training and create the cache in an online fashion, computing Δ is too costly.

5.2 Learning with Amortized Inference

When training a structured prediction model, an inference engine repeatedly solves (2.16) for every training instance. This step is, in general, computationally expensive. Although ways to reduce

[3] Srikumar et al. (2012) proposed to use $(2z^{\mathbf{p}}_{j,a} - 1)(c^{\mathbf{p}}_{j,a} - c^{\mathbf{q}}_{j,a}) \leq \epsilon, \quad \forall j, a$ for approximate amortized inference without providing a guarantee. Applying a similar derivation, we can prove that $f^{\mathbf{q}}(\boldsymbol{z}^q) - f^{\mathbf{p}}(\boldsymbol{z}^p) \leq 2\epsilon N$ under their condition, where N is the number of output variables.

the number of inference calls during training have been studied, they often treat this inference step as a black box, and overlook the possibility of amortizing the inference step. In the following, we present **AI-DCD**, an **A**mortized **I**nference framework for **D**ual **C**oordinate **D**escent method that accelerates the training process by amortizing the inference problems. We also apply the same framework to Structured Perceptron and refer to it as **AI-SP**(**A**mortized **I**nference for **S**tructured **P**erceptron).

5.2.1 Structured SVM with Amortized Inference

We start with incorporating amortized inference technique in training a structured SVM model with dual coordinate descent (DCD) (Chang and Yih, 2013) described in Chapter 2. Starting from an initial cache $\Omega = \Omega_0$, AI-DCD maintains a cache that stores the augmented-loss inference problems involved in the training process and their corresponding solution. The training process generates a sequence of models $\boldsymbol{w}^0, \boldsymbol{w}^1, \cdots, \boldsymbol{w}^*$ and alternates between two phases 1) the inference phase and 2) the model update phase. The procedure is described in Algorithm 5.

Inference Phase. In the inference phase, AI-DCD loops over each training instance and chooses active variables based on Eq. (2.16). We pose Eq. (2.16) as an ILP problem $\mathbf{q}$. Then for each entry $\mathbf{p} \in [Q]$ in the cache, if condition (5.6) holds, we assign $\boldsymbol{y}_{\mathbf{p}}$ as the solution to $\mathbf{q}$. Here $[Q]$ is the set of all inference problems with the same feasible set as problem $\mathbf{q}$. Otherwise, we solve $\mathbf{q}$ by calling the inference engine. Once we obtain the solution to $\mathbf{q}$, we add $\boldsymbol{y}_{\mathbf{q}}$ to the cache for future iterations. This way, we populate the solution cache in an online fashion during training.

To reduce the overhead of verifying Eq. (5.6), we partition the problems in $[Q]$ into groups based on their optimal solution. When solving $\mathbf{q}$, we first find

$$\boldsymbol{y}' = \underset{\boldsymbol{y} \in \{\boldsymbol{y}_{\mathbf{p}} | \mathbf{p} \in [Q]\}}{\arg\max} \quad \text{objective of } \mathbf{q}. \tag{5.7}$$

and then we only check cached problems whose optimal solution is $\boldsymbol{y}'$. This approach is related to the caching heuristic in (Joachims et al., 2009) (see later section).

Model Update Phase. In the model update phase, we sequentially visit $\alpha_{i,\boldsymbol{y}} \in \mathcal{A}$ and update

Algorithm 5 A Dual Coordinate Descent Method with Amortized Inference

$\boldsymbol{w} \leftarrow \mathbf{0}, \boldsymbol{\alpha} \leftarrow \mathbf{0}, \mathcal{A} = \emptyset$, and an initial cache $\Omega = \emptyset$
while stopping conditions are not satisfied **do**
  for all $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ (loop over each instance) **do**
    $\bar{\boldsymbol{y}} \leftarrow$ InferenceSolver$(\mathbf{q}, \Omega, \bar{\epsilon})$, where $\mathbf{q}$ be the ILP of Eq. (2.16) with $\boldsymbol{w}, \boldsymbol{x}_i, \boldsymbol{y}_i$.
    Add $(\bar{\boldsymbol{y}}, \mathbf{q})$ to Ω.
    if $\alpha_{i,\bar{\boldsymbol{y}}} \notin \mathcal{A}$ and $-\nabla D(\boldsymbol{\alpha})_{i,\bar{\boldsymbol{y}}} > \delta$, holds **then**
      $\mathcal{A} \leftarrow \mathcal{A} \cup \{\alpha_{i,\bar{\boldsymbol{y}}}\}$.
    end if
  end for
  while inner stopping conditions are not satisfied **do**
    update $\alpha_{i,\boldsymbol{y}}$ and $\boldsymbol{w}$ by Eq. (5.8) for $\alpha_{i,\boldsymbol{y}} \in \mathcal{A}$.
  end while
end while
return $\boldsymbol{w}$

Ensure: InferenceSolver$(\mathbf{q}, \Omega, \bar{\epsilon})$:
for all $\mathbf{p} \in \Omega$ and $\mathbf{p} \in [Q]$, where $[\cdot]$ is the set of all inference problems with same feasible set. **do**
  if condition (5.6) holds (with ϵ set to be $\bar{\epsilon}$) **then**
    return $\boldsymbol{y}_{\mathbf{p}}$
  end if
end for
return $\boldsymbol{y}_{\mathbf{q}}$ by solving $\mathbf{q}$ using an ILP solver.

the model based on solving a one-dimensional sub-problem:

$$\begin{aligned} \bar{d}_{i,\boldsymbol{y}} &= \arg\min_{d} D(\boldsymbol{\alpha} + d\boldsymbol{e}_{i,\boldsymbol{y}}) \quad \text{s.t.} \quad \alpha_{i,\boldsymbol{y}} + d \geq 0 \\ \alpha_{i,\boldsymbol{y}} &\leftarrow \alpha_{i,\boldsymbol{y}} + \bar{d}_{i,\boldsymbol{y}}, \quad \text{and } \boldsymbol{w} \leftarrow \boldsymbol{w} + \bar{d}_{i,\boldsymbol{y}} \phi(\boldsymbol{y}, \boldsymbol{y}_i, \boldsymbol{x}_i). \end{aligned} \tag{5.8}$$

Note that we maintain $\mathcal{A}$ throughout the optimization process. Therefore, once an $\alpha_{i,\boldsymbol{y}}$ is added into $\mathcal{A}$, it will be updated at every iteration. To maintain a small size of $\mathcal{A}$, we remove bounded dual variables using a shrinking strategy described in (Chang et al., 2013b).

Stopping Condition. We follow (Tsochantaridis et al., 2005; Hsieh et al., 2008) to stop the training process if no candidate structure is added during current iteration and if the $\|\cdot\|_\infty$ norm of the projected gradient associated with the α variables in the working set is bounded by δ.

Analysis for AI-DCD. We use the analyses presented in (Finley and Joachims, 2008). They consider two types of approximations for a combinatorial optimization problem. An under-generating method (e.g., greedy search) finds a suboptimal feasible solution, while an over-generating method

(e.g, LP relaxation) obtains an optimal solution to a superset of the feasible set. Our approximate inference scheme is in the first category. We prove the following theorem.

Theorem 9. *AI-DCD stops after adding* $O(\frac{l\Delta^2}{\delta^2}(R^2C+1))$ *candidate structures to the working set, where* $R = \max_{i,y} \|\phi(\boldsymbol{y}, \boldsymbol{y}_i, \boldsymbol{x}_i)\|$*,* l *is the number of instances,* δ *is defined in* (2.17)*, and* Δ *is the upper bound of* $\Delta(y_i, y)$*. When the optimization process stops, the empirical risk is bounded.*

Proof Sketch. The polynomial bound of the working set size is independent of the quality of the inference solution. Furthermore, the risk bound can be derived by plugging-in $\rho = 1/(1 + M\epsilon)$ in Theorems 2 and Theorems 3 in (Finley and Joachims, 2008).[4]

Getting Exact Learning Model. According to Theorem 9, if we verify condition (5.6) with $\epsilon = 0$, the solution to Eq. (2.13) is exact. Then, Algorithm 5 converges to the optimal solution of Eq. (2.13) as a standard DCD method (Chang and Yih, 2013). When $\epsilon > 0$, we only obtain an approximate solution. However, by adapting ϵ along iterations, AI-DCD can provably converge to the optimum of (2.13). The following theorem provides the conditions:

Theorem 10. *For* $T, \tau \in \mathbb{N}$*, the model generated by Algorithm 5 converges to the optimal solution of Eq.* (2.13) *if exact inference has been applied at least once every* τ *successive inference calls after iteration* T*.*

It is trivial to see that the model is optimal if exact inference is called every time after iteration T, because the objective function of Eq. (2.14) is convex. However, we relax the condition by only requiring an exact inference call every τ successive inference calls. This can be derived from the fact that the inference procedure is only involved in selecting αs into the working set. A non-zero α in the optimal solution will be selected into the working set eventually via the exact inference call if its projected gradient is larger than 0. Note that, the dual objective function value (2.14) is monotonically non-increasing whether the inference is exact or not.

5.2.2 Structured Perceptron with Amortized Inference

The same amortized inference technique described above can be applied to Structured Perceptron (Collins, 2002). The Structured Perceptron algorithm sequentially visits training examples

[4]Finley and Joachims (2008) analyze a cutting plane method with approximate inference by considering only one example and show that this result can be generalized to multiple examples. See Finley and Joachims (2008) for details.

and solves an inference problem (2.10) on an example $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ with the current model $\boldsymbol{w}$. The updates in the model are based on the best structured output $\bar{\boldsymbol{y}}$:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(\phi(\boldsymbol{x}_i, \boldsymbol{y}_i) - \phi(\boldsymbol{x}_i, \bar{\boldsymbol{y}})),$$

where η is a learning rate. Similar to AI-DCD, we can maintain a cache during the training; hence the previous inference problems can be used to amortize later ones.

Kulesza and Pereira (2008) prove that Structured Perceptron with an over-generating inference still enjoys the mistake bound guarantees. However, for an under-generating method, there is no guarantee of convergence even if the data is separable. Note that using a violation-fixing technique, we can still maintain the mistake bound guarantees (Huang et al., 2012). We leave it for future work.

5.3 Experiments and Results

In this section, we demonstrate the results on an entity-relation extraction and a multi-label classification task. Our experiments aim to show the following properties: (1) the amortization techniques significantly reduce the number of inference solver calls, and hence the training time is reduced; (2) with an adaptation on ϵ, the model converges to the same objective function value; (3) with the help of pre-cached examples, we achieve a higher amortization ratio and further speed up the training process.

Entity and Relation Recognition. The entity-relation extraction task involves the simultaneous identification of entities in a given text and the relations between them. We have introduced the task in Section 4.3

Multi-label classification Multi-label classifier predicts a set of proper labels for each instance. When label correlation is modeled, the learning and inference is often intractable. We use a dataset, scene[5], with 6 labels and 1,211 instances to demonstrate the performance of our method.

[5]Available at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets`

Entity-Relation Extraction Task

Method	ϵ	Model Type	% Solver Calls	Inf. in Training		Dual Obj.	Performance	
				Time	Speedup		Ent F1	Rel F1
`Structured SVM`								
Baseline	-	Exact	100%	293	1	895.31	87.69	47.55
AI-DCD	0	Exact	77%	274	1.1x	895.31	87.69	47.55
AI-DCD	**adapt**	**Exact**	**24%**	**162**	**1.8x**	**895.19**	**87.66**	**47.7**
AI-DCD	0.1	Approx.	54%	213	1.4x	895.31	87.69	47.55
AI-DCD	1	Approx.	29%	115	2.5x	894.73	87.66	48.01
AI-DCD	**10**	**Approx.**	**10%**	**60**	**4.8x**	**874.08**	**87.34**	**47.83**
`Structured SVM with strict stopping condition`								
Baseline		Exact	100%	1,663	1	897.51	87.76	47.40
AI-SP	1	Approx.	5%	246	6.8x	897.50	87.76	47.45
`Averaged Structured Perceptron`								
Baseline		Exact	100%	705	1	-	89.09	37.44
AI-DCD	1	Approx.	36%	319	2.2x	-	89.02	39.05

Multi-label Classification

Method	ϵ	Model Type	% Solver Calls	Inf. in Training		Dual Obj.	Hamming Loss
				Time	Speedup		
`Structured SVM`							
Baseline		Exact	100%	1,300	1	219.20	0.609
AI-DCD	adapt.	Exact	1.6%	18	72x	219.18	0.609
AI-DCD	1	Approx.	0.6%	12	108x	219.15	0.606

Table 5.1: Ratio of inference engine calls, inference time, inference speedup, final negative dual objective function value ($-D(\alpha)$ in Eq. (2.14)) and the test performance of each method. Time is in seconds. The results show that AI-DCD significantly speeds up the inference during training, while maintaining the quality of the solution. Remarkably, it requires 24% of the inference calls and 55% of the inference time to obtain an exact learning model. The reduction of time is implementation specific.

We implemented our algorithms based on a publicly available Structured SVM package[6] in JAVA and conducted experiments on a machine with Xeon E5-2440 processors. Unless otherwise stated, we show the performance of training Structured SVM model with C = 0.1 with stopping condition $\delta = 0.1$. For Perceptron, we used an averaged Perceptron implementation in JAVA and set the max number of iterations to 100. We use a commercial package, Gurobi, as a base solver to solve the ILP problems.

5.3.1 Learning with Amortized Inference

We compare the following approaches for training a structured SVM model:

[6] `http://cogcomp.cs.illinois.edu/page/software_view/JLIS`

- Baseline: the baseline method that calls an inference engine every time.
- AI-DCD ($\epsilon = \rho$): Algorithm 5 using the approximate version of the function InferenceSolver with ϵ set to be ρ. Note that when $\rho = 0$, the InferenceSolver is exact.
- AI-DCD (adaptive ϵ): We consider a simple adaptation strategy. We begin with running Algorithm 5 with $\epsilon = 10$. When the stopping condition is satisfied, we switch to use $\epsilon = 0.1$ and continue the training process. When the stopping condition is satisfied again, we change to use exact inference ($\epsilon = 0$) toward the end.

Similarly, we compare Structured Perceptron to the version with amortized inference (AI-SP).

Table 5.1 summarizes the results. Checking condition Eq. (5.6) takes only 0.04 ms, while executing an ILP engine takes 2ms to solve an inference sample in average. Therefore, when a method achieves a higher amortization ratio, it takes less time. When $\delta = 0.1$, using amortized inference with $\epsilon = 0$ reduces about 22% of calls to an inference engine. The improvement is more significant when checking validity using an approximate condition. With $\epsilon = 0.1$, more than 46% of inference calls are amortized, while the model converges to the same model as exact inferences are made. AI-DCD ($\epsilon = 1$) saves 71% of inference calls and this leads about 60% reduction of inference time during the training. Because inference is faster, AI-DCD ($\epsilon = 1$) reduces the total running time from 534 seconds to 297 seconds compared to the baseline model. Amortized inference is especially helpful in the final stage of optimization, because the model converges. For example, in the last iteration of AI-DCD ($\epsilon = 0.1$), only 12% of inferences require to call an inference engine. As a result, the amortized inference technique is particularly useful when an accurate model is required. We verify this by showing the case when running DCD solver with a strict stopping condition ($\delta = 0.01$, maximal iteration=300), AI-DCD ($\epsilon = 1$) speed up the inference by 6.8 times. When ϵ is bigger, the model starts deviating from the optimum. However, this can be remedied by using an adaptive ϵ. Using an adaptive ϵ saves 45% of inference time, while the convergence is guaranteed. Our method is not sensitive to the choice of ϵ. The first 5 rows in Table 5.1 show that using $\epsilon \leq 1$ speedups the baseline, while obtaining a solution close to the optimum.

In Theorem 8, we show that the quality of the inference is controlled by ϵ and M as defined in Eq. (5.5). Empirically, when $\epsilon = 0.1$, M is 9.4. Therefore, a 0.5-approximate solution is guaranteed. However, Theorem 8 is not tight. We observe that the worst approximation ratio is

actually 0.9996 and our inference method almost always returns an optimal solution. Even when ϵ is large ($\epsilon = 10$), we can obtain an 0.97-approximate solution on average during the learning process, after 5 outer iterations of DCD.

As we mentioned in the previous section, 1-slack SVM (Joachims et al., 2009) uses $\boldsymbol{y}'$ in Eq. (5.7) to generate a new cutting-plane. A similar heuristic can be applied in DCD. If $\boldsymbol{y}'$ is satisfied with Eq. (2.17), we can add $\boldsymbol{y}'$ into $\mathcal{A}$ and update $\boldsymbol{w}$ based on $\alpha_{i,\boldsymbol{y}'}$ without solving inference. This approach seems plausible, but does not work well in our setting. It still requires 96.3% of the inference engine calls when $\delta = 0.1$. The main reason is that after a few iterations, most αs associated with the optimal solution of instances have been added into $\mathcal{A}$. However, DCD updates all $\alpha \in \mathcal{A}$ at each loop; therefore $\alpha \in \mathcal{A}$ are unlikely to be satisfied with Eq. (2.17). As a result, this caching method does not save inference engine calls. In contrast, the amortized inference method verifies the optimality of a cached solution, and can reduce inference calls in this case.

Table 5.1 also demonstrates the effectiveness of applying amortized inference in the Structured Perceptron model. As shown in the table, most inference problems involved in training can be amortized. As a result, AI-SP with $\epsilon = 1$ significantly reduces the running time, while achieving almost the same test performance.

Multi-label classification. We further ask a question: what other structured problems can use amortized learning? We believe that problems which involve complex output structures but less output variables will benefit most from our framework. To empirically prove this hypothesis, we apply our method on a multi-label classification problem. Following Finley and Joachims (2008) and Chang et al. (2013c), we model the multi-label classification using a fully connected pairwise Markov random field. This creates a complex structured prediction problem, where the inference is generally intractable. The last block in Table 5.1 shows the results. Both the exact (adaptive ϵ) and approximate ($\epsilon = 1$) versions of AI-DCD perform extremely well. That is because the number of variables involved in the ILP formulation of the pairwise Markov random field is small[7], and the condition 5.6 is more likely to be satisfied. Moreover, the intractable nature of the problem makes the inference engine slow. Therefore, our model speeds up the inference involved in training by an order of 2. The amortized inference removes the bottleneck in the training. When a raw solver

[7]For a multi-label problem with k labels, the number of variables of the ILP formulation is $k + C(k, 2)$. In our case, k = 6.

is used, 71% of the training time is spent on solving the inference problems. When an amortized inference solver is used, the ratio of the inference time to the total running time reduces to 2%.

Note that the above experiment uses an exact inference solver. The raw solver can be replaced by any approximate inference solver, because checking condition (5.6) is independent of solver type. The amortized inference technique can be applied in other multi-label classification frameworks.

5.3.2 Cross Validation using Amortization

Our framework is also applicable in situations such as cross-validation, model selection, and feature engineering, where we have to run a learning algorithm with different parameters repeatedly. In the following, we demonstrate our methods when performing a 5-fold cross validation for picking the best regularization parameter C for structured SVM on the entity-relation recognition task. We choose C from $\{0.01, 0.5\ 0.1, 0.5, 1\}$, running 5 folds for each, for a total of 25 runs. δ is set to 0.1 for all folds.

We consider two caching policies. In the first, we reset the cache every 5 runs, so the cross validation for each C starts with an empty cache (we call this *cache resetting*). In the second setting, we do not reset the cache during any of the runs, thereby caching the inference problems in all the 25 runs (we call this *cache forwarding*). We compare both these policies with the baseline of using no cache.

The total number of solver calls for the baseline is 18.9M. With cache-resetting, this number drops to 4.4M for AI-DCD ($\epsilon = 1$). Furthermore, with cache forwarding, the number of total solver calls for AI-DCD ($\epsilon = 1$) reduces to 3.3M. When we use adaptive ϵ, the number of solver calls reduces from 12.5M for cache-resetting to 7M for cache forwarding. The reduction in solver calls is smaller when using adaptive ϵ than when $\epsilon = 1$, because the adaptive scheme does exact inference periodically in an attempt to converge to the optimal solution, thereby taking more iterations.

We noticed that the amortization suffers due to resetting the cache, and therefore the amortization ratio drops in the initial folds. We remedy this by using cache forwarding to accumulate the cache. Indeed, using an accumulated cache increases the average amortization ratio from 0.46 to 0.64 in AI-DCD ($\epsilon = 1$). The fact that problems cached for a different C aid in amortizing future problems also suggests that if we need to run the learning process multiple times (for parameter

tuning, feature selection etc.), we can benefit by populating the cache in advance. Although we performed this experiment in a serial fashion, this is not a restriction imposed by our amortization framework. Parameter tuning can be done in parallel. In principle, the cache can be shared among several threads or cluster nodes.

5.4 Conclusion

This chapter presents a novel approach to training structured predictors, by amortizing the inference step. It builds on the observation that a large number of calls to an inference engine are needed in the course of training a structured predictor, but even when the inference objectives are different, it is often the case that they produce identical solutions. Exploiting a theory that efficiently recognizes whether two inference objectives will produce the same solution, we show that training structured predictors can be accelerated significantly without any performance degradation.

The proposed framework is general for several reasons. First, the formulation of the inference step as an ILP problem is extremely general. Many inference problems in natural language processing, vision and other fields can be cast as ILP problems (Roth and Yih, 2004; Clarke and Lapata, 2006; Riedel and Clarke, 2006). In fact, all discrete MPE problems can be cast as ILPs (Roth and Yih, 2004; Sontag, 2010). Second, our framework is independent of the inference engine used to solve the inference problems, but it can maintain the exactness (or approximation) guarantees of the solver chosen by the user. Third, amortized inference can be plugged into any learning framework which requires solving inference repeatedly. (e.g., (Joachims et al., 2009; Lacoste-Julien et al., 2013)).

Chapter 6

Supervised Clustering on Steaming Data

In this chapter, we show that the ideas of selection can be applied in training metrics for clustering algorithms. We particularly focus on situations where clustered items arrive as a data stream and assume that it is sufficient to only consider the previous items when clustering an item. We call this setting online clustering setting. We will show that algorithms that take this streaming property into account achieve better performance and are more efficient.

To model this streaming property, in (Chang et al., 2013a), we presented a Latent Left-Linking Model (L^3M), where the score of a cluster is estimated by some selecting pairs of items within that cluster. Using these selecting pairs, L^3M admits efficient inference and learning. Moreover, L^3M is able to capture the underlying inherent structure that the items in clusters are presented in an order. In the following, we take coreference resolution as an example to describe the L^3M model.

We first provide an overview introduction to coreference resolution. We then show the challenges and the proposed L^3M model.

6.1 Coreference Resolution

Coreference resolution is a fundamental NLP task (Ng, 2010). From an NLP perspective, coreference resolution is largely composed of two components:

- *mention detection*, i.e., identifying boundaries of relevant denotative noun phrases (e.g. 'Russian counterpart', 'his')
- *coreference clustering*, i.e. clustering these mentions, so that two mentions share the same cluster if and only if they refer to the same entity (see example in Section 1.1.2).

Almost all the existing coreference systems first detect the mentions in the article and pipeline the identified mentions into the coreference clustering component. In this chapter, we will focus on the coreference clustering part of the pipeline, assuming the correct mention boundaries are given. Coreference clustering, from a machine learning perspective, is an online supervised clustering problem, where a clustering function learns to group mentions that arrive one by one in a predefined order. To define the clustering function, most machine learning approaches learn a scoring function to estimate the compatibility between two mentions or two sets of previously clustered mentions. Then, a decoding algorithm is designed to aggregate these scores and find an optimal clustering assignment.

There are two major ways to learn the aforementioned pairwise similarity function. Several systems developed over the past decade consider decoupling the task of learning the pairwise similarity function from the clustering of items with the similarity function (Soon et al., 2001; Ng and Cardie, 2002; Bengtson and Roth, 2008). They first learn the similarity function based on a binary classification method. Then they derive greedy algorithms for clustering the mentions. These approaches are simple and effective. However, the decoupling approaches may lose information during the training, resulting suboptimal performance. We will show an example in Section 6.2. In contrast, global learning methods formulate the learning as a structured prediction problem and treat the similarity function as a local prediction function. These approaches allow learning the pairwise similarity function with respect to the clustering algorithm, and often have better performance. However, if we model the coreference resolution problem as a structured prediction problem with a clustering structure, the inference involves solving a correlation clustering problem, which is known to be NP-hard (Garey and Johnson, 1990). Therefore, there is a need to design an efficient global learning algorithm for coreference resolution.

In Section 6.3, we will show a novel and principled machine learning framework that pushes the state-of-the-art while operating at mention-pair granularity. We present two models — the Latent Left-Linking Model (L^3M), and its version augmented with domain knowledge-based constraints, the Constrained Latent Left-Linking Model (CL^3M). L^3M admits efficient inference, linking each mention to a previously occurring mention to its left, much like the existing best-left-link inference models (Ng and Cardie, 2002; Bengtson and Roth, 2008). However, unlike previous best-link

techniques, learning in our case is performed jointly with decoding — we present a novel latent structural SVM approach, optimized using a fast stochastic gradient-based technique. Furthermore, we present a probabilistic generalization of L^3M that is more expressive in that it is capable of considering mention-entity interactions using scores at the mention-pair granularity. We augment this model with a temperature-like parameter (Samdani et al., 2012a) which provides additional flexibility. CL^3M augments L^3M with knowledge-based constraints (*a la* (Roth and Yih, 2004) and (Denis and Baldridge, 2007)). This capability is very desirable as shown by the success of the rule-based deterministic approach of Raghunathan et al. (2010) in the CoNLL shared task 2011 (Pradhan et al., 2011). In our case, domain-specific constraints are incorporated into learning and inference for L^3M in a straightforward way. CL^3M scores a mention clustering by combining the score of underlying L^3M model with the score from a set of constraints.

Overall, the main contribution of our approach is a principled machine learning model operating at mention-pair granularity, using easy to implement constraint-augmented inference and learning, that yields the best results at the time when this work is done on end-to-end coreference resolution on Ontonotes-5.0 (Pradhan et al., 2012).

In the following, we will first describe the pairwise mention scoring framework and previous attempt for solving the coreference resolution problem. Then we will discuss the latent left-linking models.

6.2 Pairwise Mention Scoring

Let d be a document with m_d number of mentions. Mentions are denoted solely using their indices ranging from 1 to m_d. A coreference clustering $\mathcal{C}$ for document d is a collection of disjoint sets partitioning the set $\{1, \ldots, m_d\}$. We represent $\mathcal{C}$ as a binary function with $\mathcal{C}(i, j) = 1$ if mentions i and j are coreferent, otherwise $\mathcal{C}(i, j) = 0$. Let $s(\mathcal{C}; \boldsymbol{w}, d)$ be the score of a given clustering $\mathcal{C}$ for a given document and a given pairwise weight vector $\boldsymbol{w}$. Then, during inference, a clustering $\mathcal{C}$ is predicted by maximizing the scoring function $s(\mathcal{C}; \boldsymbol{w}, d)$, over all valid (i.e. satisfying symmetry and transitivity) clustering binary functions $\mathcal{C} : \{1, \ldots, m_d\} \times \{1, \ldots, m_d\} \rightarrow \{0, 1\}$.

There are several ways to define the clustering function. In the following, we assume the score assigned to a clustering consists of local scores obtained from pairwise classifier which indicates the

compatibility of a pair of mentions. The inference routine then predicts the final clustering — a structured prediction problem — using these pairwise scores.

Specifically, for any two mentions i and j (w.l.o.g. $j < i$), we produce a pairwise compatibility score w_{ji} using extracted features $\phi(j, i)$ as

$$w_{ji} = \mathbf{w} \cdot \phi(j, i) \ , \tag{6.1}$$

where $\boldsymbol{w}$ is a weight parameter that is learned.

In the following, we present two inference techniques for coreference resolution. These clustering techniques take as input a set of pairwise mention scores over a document and aggregate them into globally consistent cliques representing entities. We investigate the traditional *Best-Link* approach and *All-Link* algorithm.

6.2.1 Best-Link

Best-Link is a popular approach for coreference resolution. For each mention, it considers the best mention on its left to connect to (best according the pairwise score w_{ji}) and creates a link between them if the pairwise score is above some threshold. Although its strategy is simple, Bengtson and Roth (2008) show that with a careful design, it can achieve highly competitive performance.

Inference: The inference problem in *Best-Link* model can be solved by the greedy algorithm mentioned above. However, here we provide an integer linear programming (ILP) formulation of *Best-Link* inference in order to present both of our inference algorithms within the same framework. Given a pairwise scorer $\mathbf{w}$, we can compute the compatibility scores — w_{ji} from Eq. (6.1) — for all mention pairs j and i. Let y_{ji} be a binary variable, such that $y_{ji} = 1$ *only if* j and i are in the same cluster (i.e., $y_{ji} = 1 \Rightarrow C(j, i) = 1$). For a document d, *Best-Link* solves the following ILP

formulation:

$$\begin{aligned} \arg\max_{y} \quad & \sum_{j,i} w_{ji} y_{ji} \\ \text{s.t.} \quad & \sum_{j<i} y_{ji} \leq 1 \ \forall i, \\ & y_{ji} \in \{0,1\}. \end{aligned} \tag{6.2}$$

Eq. (6.2) generates a set of connected components and all the mentions in each connected component constitute an entity. Note that the objective function in Eq. (6.2) defines the clustering score function $s(C; \boldsymbol{w}, d)$.

Learning: Bengtson and Roth (2008) describe an approach that decouples the learning of $\boldsymbol{w}$ from the clustering strategy. They learn the pairwise scoring function by training a binary classifier, where

- positive examples: for each mention i, we construct a positive example (j, i), where j is the closest preceding mention in i's equivalence class, and
- negative examples: all mention pairs (j, i), where j is a preceding mention of i and j, i are not in the same class.

Although this decoupling approach is effective, some information might be lost during the training. For example, consider the following paragraph:

> [**Christopher Robin**] is alive and well. ... When Chris was three years old, his father wrote a poem about him. The poem was printed in a magazine for others to read. [**Mr. Robin**] then wrote a book. ....

In the decoupling approach, the mention pair [**Christopher Robin**] and [**Mr. Robin**] forms a negative sample. However, it is very hard to identify that they are different from the mentions' surface strings and contexts. However, it is relatively easy to see that the pronoun "his" and [**Mr. Robin**] are coreferred, because they are close to each other in the paragraph and their contexts are similar. The *Best-Link* inference algorithm, which produces a global clustering, takes

all mentions into consideration, whereas this information is not available during learning in the decoupling framework, resulting in sub-optimal performance.

6.2.2 All-Link

The *All-Link* inference approach scores a clustering of mentions by including all possible pairwise links in the score. It is also known as correlation clustering (Bansal et al., 2002) and has been applied to coreference resolution in the form of supervised clustering (Mccallum and Wellner, 2003; Finley and Joachims, 2005).

Inference: As we mentioned in Section 2.2, for a document d, *All-Link* inference finds a clustering $A\text{ll-Link}(d; w)$ by solving the following ILP problem (Eq. (2.11))

$$\begin{aligned} \arg\max_{y} \quad & \sum_{j,i} w_{ji} y_{ji} \\ \text{s.t} \quad & y_{jk} \geq y_{ji} + y_{ik} - 1 \ \forall i, j, k, \\ & y_{ji} \in \{0, 1\}, \end{aligned}$$

where the binary variable $y_{ji} = 1$ *if and only if* mentions i and j are in the same cluster $y_{ji} \Leftrightarrow C(j, i) = 1$. The inequality constraints in Eq. (2.11) enforce the transitive closure of the clustering, and the objective function represents $s(C; \boldsymbol{w}, d)$. The solution of Eq. (2.11) is a set of cliques, and the mentions in the same cliques corefer.

Learning: We present a structured perceptron algorithm, which is similar to supervised clustering algorithm (Finley and Joachims, 2005) to learn w. Note that as an approximation, it is certainly possible to use the weight parameter learned by using, say, averaged perceptron over positive and negative links. The pseudocode is presented in Algorithm 6.

For the *All-Link* clustering, we drop one of the three transitivity constraints for each triple of mention variables. Similar to Pascal and Baldridge (2009), we observe that this improves accuracy — the reader is referred to Pascal and Baldridge (2009) for more details.

When the number of mentions is small, the inference problem in Eq. (2.11) can be solved by an ILP solver (e.g., Gurobi Optimization, Inc. (2012)). However, if the number of mentions is

Algorithm 6 Structured Perceptron like learning algorithm for All-Link inference

Given: Annotated documents D and initial weight $\boldsymbol{w}_{init}$
Initialize $\boldsymbol{w} \leftarrow \boldsymbol{w}_{init}$
for Document d in D **do**
  Clustering $y \leftarrow A$ll-Link$(d; w)$
  for all pairs of mentions u and v **do**
    $\mathcal{I}^1(j,i) = [j, i \text{ coreferent in } D]$
    $\mathcal{I}^2(j,i) = [y(j) = y(i)]$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \left(\mathcal{I}^1(j,i) - \mathcal{I}^2(j,i)\right) \phi(j,i)$
  end for
end for
return $\boldsymbol{w}$

large, there is no efficient way to solve Eq. (2.11). Moreover, the *All-Link* approach does not take the streaming property of the online clustering problems into account, leading to a suboptimal model.

6.3 Latent Left-Linking Model

In this section, we describe our Constrained Latent Left-Linking Model (CL^{3}M). CL^{3}M is inspired by a few ideas from the literature: (a) the popular *Best-Left-Link* inference approach to coreference (Ng and Cardie, 2002; Bengtson and Roth, 2008), and (b) the injection of domain knowledge-based constraints for structured prediction (Roth and Yih, 2005; Clarke and Lapata, 2006; Chang et al., 2007b; Ganchev et al., 2010; Koo et al., 2010; Pascal and Baldridge, 2009). We first introduce our Left-Linking Model (L^{3}M) and then describe how to inject constraints into our model.

We present our inference algorithm that is inspired by the best-left-link approach. In particular, the score $s(\mathcal{C}; d, \boldsymbol{w})$ is expressed based on the model that each mention links to an antecedent mention (to its left) with the highest score (as long as the score is above some threshold, say, 0). So we write down s as

$$s(\mathcal{C}; d, \boldsymbol{w}) = \sum_{i=1}^{m_d} \max_{0 \le j < i, \mathcal{C}(i,j)=1} \boldsymbol{w} \cdot \phi(j,i) \ . \tag{6.3}$$

In order to simplify the notation, we have introduced a dummy mention with index 0, which is to the left (i.e. appears before) all the mentions and has $w_{0i} = 0$ for all *actual* mentions $i > 0$. For a

given clustering $\mathcal{C}$, if a mention i is not co-clustered with any previous actual mention j, $0 < j < i$, then we assume that i links to 0 and $\mathcal{C}(i, 0) = 1$. In other words, $C(i, 0) = 1$ *iff* i is the first actual item of a cluster in $\mathcal{C}$. However, such an item i is **not** considered to be co-clustered with 0 and for any valid clustering, item 0 is always in a singleton dummy cluster, which is eventually discarded. s is maximized exactly by the best-left-link inference, as it maximizes individual left link scores and creation of one left-link does not affect creation of other left-links.

6.3.1 Learning

We use a max-margin approach to learn $\boldsymbol{w}$. We are given a training set D of documents where for each document $d \in D$, $\mathcal{C}_d$ refers to the annotated ground truth clustering. Then we learn $\boldsymbol{w}$ by minimizing

$$L(\boldsymbol{w}) = \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{|D|}\sum_{d\in D}\frac{1}{m_d}\Big(\max_{\mathcal{C}}\big(s(\mathcal{C}; d, \boldsymbol{w}) + \Delta(\mathcal{C}, \mathcal{C}_d)\big) - s(\mathcal{C}_d; d, \boldsymbol{w})\Big) \ , \tag{6.4}$$

where $\Delta(\mathcal{C}, \mathcal{C}_d)$ is a loss function. In order to achieve tractable loss-augmented minimization — something not possible with standard loss functions (e.g. B^3 (Bagga and Baldwin, 1998)) used in coreference — we use a decomposable loss function that just counts the number of mention pairs on which $\mathcal{C}$ and $\mathcal{C}_d$ disagree: $\Delta(\mathcal{C}, \mathcal{C}_d) = \sum_{i,j=0,j<i}^{m_d} \mathbb{I}_{\mathcal{C}(i,j)=\mathcal{C}_d(i,j)}$, where $\mathbb{I}$ is a binary indicator function. With this form of loss function and using the form of scoring function from Eq. (6.3), we can write $L(\boldsymbol{w})$ as

$$\frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{|D|}\sum_{d\in D}\frac{1}{m_d}\sum_{i=1}^{m_d}\Big(\max_{0\le j<i}\big(\boldsymbol{w}\cdot\phi(j,i) + \delta(\mathcal{C}_d, i, j)\big) - \max_{0\le j<i, C(i,j)=1}\big(\boldsymbol{w}\cdot\phi(j,i)\big)\Big) \ , \tag{6.5}$$

where $\delta(\mathcal{C}_d, i, j) = 1 - \mathcal{C}_d(i, j)$ is the loss-based margin that is 1 if i and j are not coreference in $\mathcal{C}_d$, else 0. In the above objective function, the left-links remain latent while we get to observe the clustering. This objective function is related to latent structural SVMs (Yu and Joachims, 2009). However (Yu and Joachims, 2009) use a spanning tree based latent structure which does not have the left-to-right directionality we exploit. We can minimize the above function using Concave Convex Procedure (Yuille and Rangarajan, 2003), which is guaranteed to reach the local

minima. However, such a procedure is costly as it requires doing inference on all the documents to computer a single gradient update. Consequently, we choose a faster stochastic sub-gradient descent (SGD). Since $L(\boldsymbol{w})$ in Eq. (6.5) decomposes not only over training documents, but also over individual mentions in each document, we can perform SGD on a per-mention basis. The stochastic sub-gradient w.r.t. mention i in document d is given by

$$\begin{aligned} \nabla L(\boldsymbol{w})_d^i &\propto \phi(j', i) - \phi(j'', i) + \lambda \boldsymbol{w}, \quad \text{where} \\ j' &= \arg\max_{0 \le j < i} (\boldsymbol{w} \cdot \phi(j, i) + 1 - \mathcal{C}_d(i, j)) \\ j'' &= \arg\max_{0 \le j < i, C(i,j)=1} \boldsymbol{w} \cdot \phi(j, i) \end{aligned} \tag{6.6}$$

While SGD has no theoretically convergence guarantee, it works excellently in our experiments.

6.3.2 Incorporating Constraints

Next, we show how to incorporate domain knowledge-based constraints into L^{3}M and generalize it to CL^{3}M. In CL^{3}M, we obtain a clustering by maximizing a constraint-augmented scoring function f given by

$$s(\mathcal{C}; d, \boldsymbol{w}) + \sum_{p=1}^{n_c} \rho_p \psi_p(d, \mathcal{C}), \tag{6.7}$$

where the second term on the R.H.S. is the score contributed by domain specific constraints $\psi_1, \ldots, \psi_{n_c}$ with their respective scores $\rho_1, \ldots, \rho_{n_c}$. In particular, $\psi_p(d, \mathcal{C})$ measures the extent to which a given clustering $\mathcal{C}$ satisfies the p^{th} constraint (e.g., a constraint can count the number of proper name mentions that have high lexical similarity.) Note that this framework is general and can be applied to inject mention-to-cluster or cluster-to-cluster level constraints. However, for simplicity, we consider only constraints between mention pairs in this thesis. This allows us derive fast greedy algorithm to solve the inference problem. The details of our constraints are presented in Section 6.6.

All of our constraints can be categorized into two groups: "must-link" and "cannot-link". "Must-link" constraints encourage a pair of mentions to connect, while "cannot-link" constraints discour-

age mention pairs from being linked. Consequently, the coefficients ρ_p associated with "must-link" constraints are positive while ρ_p for "cannot-link" constraints are negative. In the following, we briefly discuss how to solve the inference problem with these two types of constraints.

We slightly abuse notations and use $\psi_p(j,i)$ to indicate the p^{th} constraint on a pair of mentions (i,j) $\psi_p(j,i)$ is a binary function that is 1 *iff* two mentions i and j satisfy the conditions specified in constraint p. Chang et al. (2011) shows that best-left-link inference can be formulated as an ILP problem. When we add constraints, the ILP becomes:

$$\begin{aligned} \underset{B,C\in\{0,1\}}{\arg\max} \quad & \sum_{i,j:j<i} w_{ji}B_{ji} + \sum_{i,j} \rho_p \psi_p(j,i) C_{ij} \\ \text{s.t} \quad & C_{kj} \geq C_{ij} + C_{ki} - 1, \ \forall i,j,k, \\ & \sum_{j=0}^{i-1} B_{ji} = 1, \ \forall i \\ & B_{ji} < C_{ji}, C_{ji} = C_{ji}, \forall i,j, \end{aligned} \tag{6.8}$$

where $C_{ij} \equiv C(i,j)$ is a binary variable indicating whether i and j are in the same cluster or not and B_{ji} is an auxiliary variable indicating the best-left-link for mention i. The first set of inequality constraints in (6.8) enforces the transitive closure of the clustering. The constraints $B_{ji} < C_{ji}, \forall i,j$ enforce the consistency between these two sets of variables.

One can use an off-the shelf solver to solve Eq. (6.8). However, when the absolute values of the constraint scores ($|\rho_p|$) are high (the hard constraint case), then the following greedy algorithm approximately solves the inference efficiently. We scan the document from left-to-right (or in any other arbitrary order). When processing mention i, we find

$$j^* = \underset{j<i}{\arg\max} \quad w_{ji} + \sum_{k:\hat{C}(k,j)=1} \sum_{p} \rho_p \psi_p(k,i), \tag{6.9}$$

where $\hat{C}$ is the current clustering obtained from the previous inference steps. Then, we add a link between mention i and j^*. Then, the rest of inference process is the same as in the original best-left-link inference. Specifically, this inference procedure combines the classifier score mention pair i,j, with the constraints score of all mentions currently co-clustered with j. We will provide

more discussions in Section 6.6

6.4 Probabilistic Latent Left-Linking Model

In this section, we extend and generalize our left-linking model approach to a probabilistic model, Probabilistic Latent Left-Linking Model (PL^{3}M), that allows us to naturally consider mention-to-entity (or mention-to-cluster) links. While in L^{3}M, we assumed that each mention links deterministically to the max-scoring mention on its left, in PL^{3}M, we assume that mention i links to mention j, $j \leq i$, with probability given by

$$Pr[j \leftarrow i; d, \boldsymbol{w}] = \frac{e^{\frac{1}{\gamma}(\boldsymbol{w} \cdot \phi(i,j))}}{Z_i(\boldsymbol{w}, \gamma)} \ . \tag{6.10}$$

Here $Z_i(\boldsymbol{w}, \gamma) = \sum_{0 \leq k < i} e^{\frac{1}{\gamma}(\boldsymbol{w} \cdot \phi(i,k))}$ is a normalizing constant and $\gamma \in (0, 1]$ is a constant temperature parameter that is tuned on a development set (Samdani et al., 2012a). We assume that the event that mention i links to a mention j is independent of the event that mention i' links to j' for $i \neq i'$.

Inference with PL^{3}M: Given the probability of a link as in Eq. (6.10), the probability that mention i joins an existing cluster c, $Pr[c \odot i; d, \boldsymbol{w}]$, is simply the sum of the probabilities of i linking to the mentions inside c:

$$\begin{aligned} Pr[c \odot i; d, \boldsymbol{w}] &= \sum_{j \in c, 0 \leq j < i} Pr[j \leftarrow i; d, \boldsymbol{w}] \\ &= \sum_{j \in c, 0 \leq j < i} \frac{e^{\frac{1}{\gamma}(\boldsymbol{w} \cdot \phi(i,j))}}{Z_i(d, \boldsymbol{w}, \gamma)} \ . \end{aligned} \tag{6.11}$$

Based on Eq. (6.11) and making use of the independence assumption of left-links, we follow a simple greedy clustering (or inference) algorithm: sequentially add each mention i to a previously formed cluster c^*, where $c^* = \arg\max_c Pr[c \odot i; d, \boldsymbol{w}]$. If the arg max cluster is the singleton cluster with the dummy mention 0 (i.e. the score of all other clusters is below the threshold of 0), then i starts a new cluster and is not included in the dummy cluster. Note that we link a mention to a cluster taking into account all the mentions inside that cluster, mimicking the notion of a mention-to-cluster link.

This provides more expressiveness than the Best-Left-Link inference, where a mention connects to a cluster solely based on a single pairwise link to some antecedent mention (the best-link mention) in that cluster.

The case of $\gamma = 0$: As γ approaches zero, it is easy to show that the probability $P[j \leftarrow i; d, w]$ in Eq. (6.10) approaches a Kronecker delta function that puts probability 1 on the *max-scoring mention* $j = \arg\max_{0 \leq k < i} \boldsymbol{w} \cdot \phi(i, j)$ (assuming no ties), and 0 everywhere else (Pletscher et al., 2010; Samdani et al., 2012a). Consequently, as $\gamma \rightarrow 0$, $Pr[c \odot i; d, \boldsymbol{w}]$ in Eq. 6.11 approaches a Kronecker delta function centered on the cluster containing the max-scoring mention, thus reducing to the best-link case of L^{3}M. Thus, PL^{3}M, when tuning the value of γ, is a strictly more general model than L^{3}M.

Learning with PL^{3}M We use a likelihood-based approach to learning with PL^{3}M, and first compute the probability $Pr[\mathcal{C}; d, \boldsymbol{w}]$ of generating a clustering $\mathcal{C}$, given $\boldsymbol{w}$. We then learn $\boldsymbol{w}$ by minimizing the regularized negative log-likelihood of the data, augmenting the partition function with a loss-based margin (Gimpel and Smith, 2010). We omit the details of likelihood computation due to lack of space.

With PL^{3}M, we again follow a stochastic gradient descent technique instead of CCCP for the same reasons mentioned in Sec. 6.3.1. The stochastic gradient (subgradient when $\gamma = 0$) w.r.t. mention i in document d is given by

$$\nabla LL(\boldsymbol{w})^i_d \propto \sum_{0 \leq j < i} p_j \phi(i, j) - \sum_{0 \leq j < i} p'_j \phi(i, j) + \lambda \boldsymbol{w}, \tag{6.12}$$

where p_j and p'_j, $j = 0, \ldots, i-1$, are non-negative weights that sum to one and are given by

$$\begin{aligned} p_j &= \frac{e^{\frac{1}{\gamma}(\boldsymbol{w} \cdot \phi(i,j) + \delta(\mathcal{C}_d, i, j))}}{\sum_{0 \leq k < i} e^{\frac{1}{\gamma}(\boldsymbol{w} \cdot \phi(i,k) + \delta(\mathcal{C}_d, i, k))}} \text{ and} \\ p'_j &= \frac{\mathcal{C}_d(i, j) Z_i(d, \boldsymbol{w}, \gamma)}{Z_i(\mathcal{C}_d; d, \boldsymbol{w}, \gamma)} Pr[j \leftarrow i; d, \boldsymbol{w}] \ . \end{aligned}$$

Interestingly, the above update rule generalizes the one for L^{3}M, as we are incorporating a weighted sum of all previous mentions in the update rule. With $\gamma \rightarrow 0$, the SGD in Eq. (6.6) converges to

the SGD update in L^3M (Eq. (6.6)). Finally, in the presence of constraints, we can fold them inside the pairwise link probabilities as in Eq. (6.9).

6.5 Related Work

The idea of Latent Left-linking Model (L^3M) is inspired by a popular inference approach to coreference which we call the *Best-Left-Link* approach (Ng and Cardie, 2002; Bengtson and Roth, 2008). In the best-left-link strategy, each mention i is connected to the best antecedent mention j with $j < i$ (i.e. a mention occurring to the left of i, assuming a left-to-right reading order), thereby creating a *left-link*. The "best" antecedent mention is the one with the highest pairwise score w_{ij}; furthermore, if w_{ij} is below some threshold, say 0, then i is not connected to any antecedent mention. The final clustering is a transitive closure of these "best" links. The intuition for best-left-link strategy is placed in how humans read and decipher coreference links. This strategy is vastly successful and popular in NLP (Ng and Cardie, 2002; Bengtson and Roth, 2008; Stoyanov et al., 2009). However, most works have dealt with this idea in an ad hoc fashion. For instance, Bengtson and Roth (2008) train $\boldsymbol{w}$ on binary training data generated by taking for each mention, the closest antecedent coreferent mention as a positive example, and all the other mentions in between as negative examples. The system by Chang et al. (2012a) at the CoNLL shared task 2012 on coreference resolution applies a latent structured Perceptron approach to train a best-left-link model. However, a detailed analysis of their technique is not provided. In this chapter, we formalize the learning problem of the best-left-link model as a structured prediction problem and analyze our system with detailed experiments. Furthermore, we generalize this approach by considering multiple pairwise left-links instead of just the best link, efficiently capturing the notion of a mention-to-cluster link.

Many techniques in coreference literature break away from the mention pair-based, best-left-link paradigm. Denis and Baldridge (2008) and Ng (2005) learn a local ranker to rank the mention pairs based on their compatibility. While these approaches achieve decent empirical performance, it is unclear why these are the right ways to train the model. Some techniques consider a more expressive model by using features defined over mention-cluster or cluster-cluster Rahman and Ng (2011b); Stoyanov and Eisner (2012); Haghighi and Klein (2010). For these models, the inference and learning algorithms are usually complicated. Very recently, Durrett et al. (2013) propose a

probabilistic model which enforces structured agreement constraints between specified properties of mention cluster when using a mention-pair model. This approach is very related to the probabilistic extension of our method as both the models attempt to leverage entity-level information from mention-pair features. However, our approach is simpler because it directly considers the probabilities of multiple links. Furthermore, while their model performs only slightly better than the Stanford rule based system (Lee et al., 2011), we greatly outperform the same. Most importantly, our model obtains the best end-to-end performance reported to date on OntoNotes-5.0 while still operating at mention-pair granularity. We believe that this is due to our novel and principled structured prediction framework which results in accurate (and efficient) training.

Several structured prediction techniques have been applied to coreference resolution in the machine learning literature. For example, Mccallum and Wellner (2003) and Finley and Joachims (2005) model coreference as a correlation clustering (Bansal et al., 2002) problem on a complete graph over the mentions with edge weights given by the pairwise classifier. However, correlation clustering is known to be NP Hard (Bansal et al., 2002); nonetheless, an ILP solver or an approximate inference algorithm can be used to solve this problem. Another approach proposed by Yu and Joachims (2009) formulates coreference with latent spanning trees. However, their approach has no directionality between mentions, whereas our latent structure captures the left-to-right ordering of mentions. In our experiments, we show that our technique vastly outperforms both the spanning tree and correlation clustering techniques. We also compare with Fernandes et al. (2012) and the publicly available Stanford coreference system (Raghunathan et al., 2010; Lee et al., 2011), which is the state-of-the-art rule-based system.

Finally, some research Ratinov and Roth (2012); Bansal and Klein (2012); Rahman and Ng (2011c) has tried to integrate world knowledge from web-based statistics or knowledge database into a coreference system. World knowledge is potentially useful for resolving coreference and can be injected into our system via the constraint framework. We will show an example of incorporating our system with name-entity and WordNet-based similarity metric Do et al. (2009) in Section (6.6). Including the massive amount of information from knowledge resources is not the focus of this thesis.

6.6 Experiments

In this section, we present our experiments on the two commonly used benchmark English coreference datasets — Ontonotes-5.0 (Pradhan et al., 2012) and ACE 2004 (NIST, 2004). Table 6.1 exhibits our bottom line results: CL^{3}M achieves the best result reported on Ontonotes-5.0 so far. We show that CL^{3}M performs particularly well on clusters containing named entity mentions, which are more important for many information extraction applications. In the rest of this section, after describing our experimental setting, we provide careful analysis of our algorithms and compare them to competetive coreference approaches in the literature.

6.6.1 Experimental Setup

Datasets: ACE 2004 contains 443 documents — we used a standard split of these documents into 268 training, 68 development, and 106 testing documents used by Culotta et al. (2007) and Bengtson and Roth (2008). OntoNotes-5.0 dataset, released for the CoNLL 2012 Shared Task (Pradhan et al., 2012), is by far the largest annotated corpus on coreference. It contains 3,145 annotated documents drawn from a wide variety of sources — newswire, bible, broadcast transcripts, magazine articles, and web blogs. Since the actual test data for the shared task competition was never released, we use the provided development set for testing, and split the provided training data into training and development sets (note that many of the CoNLL 2012 Shared Task papers have reported results only on the test data.)

Classifier: For each of the pairwise approach, we assume the pairwise score is given by $\boldsymbol{w} \cdot \phi(\cdot, \cdot) + t$ where ϕ are the features, $\boldsymbol{w}$ is the learned weight vector, and t is a threshold which we set to 0 during learning (as in Eq. (6.1)), but use a tuned value (tuned on a development set) during testing. For learning with L^{3}M, we do stochastic gradient descent with 5 passes over the data. Empirically, we observe that this is enough to generate a stable model. For PL^{3}M (Sec. 6.4), we tune the value of γ using the development set picking the best γ from $\{0.0, 0.2, \ldots, 1.0\}$. Recall that when $\gamma = 0$, PL^{3}M is the same as L^{3}M. We refer to L^{3}M and PL^{3}M with incorporating constraints during inference as CL^{3}M and CPL^{3}M (Sec. 6.3.2), respectively.

Metrics: We compare the systems using three popular metrics for coreference — MUC (Vilain et al., 1995), BCUB (Bagga and Baldwin, 1998), and Entity-based CEAF (CEAF$_e$) (Luo, 2005).

Following, the CoNLL shared tasks (Pradhan et al., 2012), we use the average F1 scores of these three metrics as the main metric of comparison.

Features: We build our system on the publicly available Illinois-Coref system[1] primarily because it contains a rich set of features presented in Bengtson and Roth (2008) and Chang et al. (2012a) (the latter adds features for pronominal anaphora resolution.) We also compare with the Best-Left-Link approach described by Bengtson and Roth (2008).

Constraints: We consider the following constraints in CL^{3}M and CPL^{3}M.

- SameSpan: two mentions must be linked to each other if they share the same surface text span and the number of words in the text span is larger than a threshold (set as 5 in our implementation).
- SameDetNom: two mentions must be linked to each other if both mentions start with a determiner and the similarity score between the head words measured by a [0,1] wordnet-based similarity metric.
- SameProperName: two mentions must be linked if they are both proper names and the similarity score measured by a named entity-based similarity metric, Illinois NESim[2], are higher than a threshold (set to 0.8). For a person entity we add additional rules to extract the first name, last name and professional title as properties.
- ModifierMismatch: the constraint prevents two mentions to be linked if the head modifiers are conflicted. For example, the constraint prevents "northern Taiwan" from linking to "southern Taiwan". We gather a list of mutual exclusive modifiers from the training data.
- PropertyMismatch: the constraint prevents two mentions to be linked if their properties are conflicted. For example, it prevents male pronouns to link to female pronouns and "Mr. Clinton" to link to "Mrs. Clinton" by checking the gender property. The properties we consider are gender, number, professional title and the nationality.

While the "must-link" constraints described in the chapter can be treated as features, due to their high precision nature, treating them as hard constraints (set ρ to a high value) is a safe and direct way to inject human knowledge into the learning model. Moreover, our framework allows a constraint to use information from previous decisions (such as "cannot-link" constraints). Treating

[1]The system is available at `http://cogcomp.cs.illinois.edu/page/software_view/Coref/`

[2]`http://cogcomp.cs.illinois.edu/page/software_view/NESim`

	MUC	BCUB	$CEAF_e$	AVG
Stanford	64.30	70.46	46.35	60.37
Chang et al. (2012a)	65.75	70.25	45.30	60.43
Martschat et al. (2012)	66.97	70.36	46.60	61.31
Björkelund and Farkas (2012)	67.12	71.18	46.84	61.71
Chen and Ng (2012)	66.4	71.8	**48.80**	62.30
Fernandes et al. (2012)	**69.46**	71.93	48.66	63.35
L^3M	67.88	71.88	47.16	62.30
CL^3M	69.20	**72.89**	48.67	**63.59**

Table 6.1: Performance on OntoNotes-5.0 with predicted mentions. We report the F1 scores (%) on various coreference metrics (MUC, BCUB, CEAF). The column AVG shows the averaged scores of the three. The proposed model (CL^3M) outperforms all the others. However, we observe that PL^3M and CPL^3M (see Sec. 6.4) yields the same performance as L^3M and CL^3M, respectively as the tuned γ for all the datasets turned out to be 0.

such constraints as features will complicate the learning model.

6.6.2 Performance of the End-to-End System

We compare our system with the top systems reported in the CoNLL shared task 2012 as well as with the Stanford's publicly released rule-based system (Lee et al., 2013, 2011), which won the CoNLL 2011 Shared Task (Pradhan et al., 2011). Note that all the systems use the same predicted annotations (e.g., gender prediction, part-of-speech tags, name entity tags) provided by the shared task organizers. However, each system implements its own mention detector and pipelines the identified mentions into the coreference clustering component. Moreover, different systems use a different set of features in the implementations. In order to partially control for errors on mention detection and evaluate the clustering component in our coreference system, we will present results on correct (or gold) mentions in the next section.

Table 6.1 shows the results. Only the best performing system of Fernandes et al. (2012) is better than L^3M, but it is outperformed by our system with constraints, CL^3M, by 0.24%. To our knowledge, the performance of CL^3M is the best result reported on Ontonotes-5.0 so far.

Performance on named entities: The coreference annotation in Ontonotes 5.0 includes various types of mentions. However, not all mention types are equally interesting. In particular, clusters which contain at least one proper name or a named entity mention are more important for information extraction tasks like Wikification (Mihalcea and Csomai, 2007), cross-document

coreference resolution (Bagga and Baldwin, 1998), and entity linking and knowledge based population (Ji and Grishman, 2011).

Inspired by this, we compare our system to the best systems in the CoNLL shared task of 2011 (Stanford Lee et al. (2011)) and 2012 (Fernandes Fernandes et al. (2012)) on the following specific tasks on Ontonotes-5.0.

- **ENT-C**: Evaluate the system on the clusters that contain at least one proper name mention. We generate the gold annotation and system outputs by using the gold and predicted name entity tag annotations provided by the CoNLL shard task 2012. That is, if a cluster does not include any name entity mention, then it will be removed from the final clustering.
- **PER-C**: Similar to the construction of ENT-C, we only consider clusters which contain at least one "Person (PER)" entity.
- **ORG-C**: Similar to the construction of Entity-C, we only consider clusters which contain at least one "Organization (ORG)" entity.

Typically, the clusters that get ignored in the above definitions contain only first and second person pronouns (it typically happens in transcribed discourse.) Also note that all the systems are trained with the same name entity tags, provided by the shared task organizers, and we use the same name entity tags to construct the specific clustering. Also, in order to further ensure fairness, we do not tune our system to favor the evaluation of these specific types of clusters. We choose to do so because we only have access to the system output of Fernandes.

Table 6.2 shows the results. The performance of all systems degrade when considering only clusters that contain name entity, indicating that ENT-C is actually a harder task than the original coreference resolution problem. In particular, resolving ORG coreferent clusters is hard, because names of organizations are sometimes confused with person names, and they can be referred to using a range of pronouns (including "we" and "it").) Overall, CL^3M outperforms all the competing systems on the clusters that contain at least one specific type of entity by a margin larger than that for overall coreference.

Task	Metric	Stanford	Fernandes	CL^{3}M
ENT-C	44.06	47.05	46.63	**48.02**
PER-C	34.04	36.43	37.01	**37.57**
ORG-C	25.02	26.23	26.22	**27.01**

Table 6.2: Performance on named entities for OntoNotes-5.0 data. We compare our system to Fernandes (Fernandes et al., 2012) and Stanford (Lee et al., 2013) systems.

6.6.3 Analysis on Gold Mentions

In this section, we present experiments assuming the gold mentions are given. The definition of gold mentions in ACE and Ontonotes are different because Ontonotes-5.0 excludes singleton clusters in the annotation. In addition, Ontonotes includes longer mentions; for example, it includes NP and appositives in the same mention. We compare with the publicly available Stanford (Lee et al., 2011) and IllinoisCoref (Chang et al., 2012a) systems; the system of (Fernandes et al., 2012) is not publicly available. In addition, we also compare with the following two structured prediction baselines that use the same set of features as L^{3}M and PL^{3}M.

1. **Corr-Red:** a reduced and faster alternative to the correlational clustering based approach (Finley and Joachims, 2005). We implement this as an ILP, and drop one of the three transitivity constraints for each triplet of mention variables. Similar to (Pascal and Baldridge, 2009) and (Chang et al., 2011) we observe that this slightly improves the accuracy over a pure correlation clustering approach, in addition to speeding up inference.
2. **Spanning:** the latent spanning forest based approach presented by Yu and Joachims (2009). We use the publicly available implementation provided by the authors[3] for the ACE data; since their CCCP implementation is slow, we implemented our own stochastic gradient descent version to scale it to the much larger Ontonotes data.

Table 6.3 lists the results. Although L^{3}M is simple and use only the features defined on pairwise mentions, it achieves competitive performance to the recently published results. Moreover, the probabilistic generalization of L^{3}M, PL^{3}M, achieves better performance. For example, L^{3}M with $\gamma = 0.2$ improves L^{3}M with $\gamma = 0$ by 0.7 points in ACE 2004. In particular, This shows that considering more than a one left-links is helpful. This is in contrast with the predicted mentions where $\gamma = 0$ provides the best results. We suspect that this is because noisy mentions can hurt the

[3]Available at http://www.cs.cornell.edu/ cnyu/latentssvm/

	MUC	BCUB	$CEAF_e$	AVG
ACE 2004 Gold Ment.				
All-Link-Red.	77.45	81.10	77.57	78.71
Spanning	73.31	79.25	74.66	75.74
IllinoisCoref	76.02	81.04	77.6	78.22
Stanford	75.04	80.45	76.75	77.41
Stoyanov and Eisner (2012)	**80.1**	81.8	-	-
L^3M	77.57	81.77	78.15	79.16
PL^3M	78.18	82.09	79.21	79.83
CL^3M	78.17	81.64	78.45	79.42
CPL^3M	78.29	**82.2**	**79.26**	**79.91**
Ontonotes 5.0 Gold Ment.				
All-Link-Red.	83.72	75.59	64.00	74.44
Spanning	83.64	74.83	61.07	73.18
IllinoisCoref	80.84	74.29	65.96	73.70
Stanford	82.26	76.82	61.69	73.59
L^3M	83.44	78.12	64.56	75.37
PL^3M	83.97	78.25	65.69	75.97
CL^3M	84.10	78.30	68.74	77.05
CPL^3M	**84.80**	**78.74**	68.75	**77.43**

Table 6.3: Performance on ACE 2004 and OntoNotes-5.0. All-Link-Red. is based on correlational clustering; Spanning is based on latent spanning forest based clustering (see Sec. 6.5). Our proposed approach is L^3M (Sec. 6.3) and PL^3M (sec. 6.4). CL^3M and CPL^3M are the version with incorporating constraints.

performance of PL^3M that takes into account not just the best scoring links, but also weaker links which are likely to be less reliable (more false positives.) Also, as opposed to what is reported by Yu and Joachims (2009), the correlation clustering approach performs better than the spanning forest approach. We think that this is because we compare the systems on different metrics than them and also because we use exact ILP inference for correlational clustering whereas Yu and Joachims used approximate greedy inference.

Both L^3M and PL^3M can be benefit from using constraints. However, The constraints improve only marginally on ACE 2004 data because ACE uses shorter phrases to represent mentions. Consequently, the constraints designed for leveraging information from long mention spans are less fired. Overall, the experiments show that L^3M and PL^3M perform well on modeling coreference clustering.

	MUC	BCUB	$CEAF_e$	AVG
L^3M	67.88	71.88	47.16	62.30
+SameSpan	68.27	72.27	47.73	62.75
+SameDetNom	68.79	72.57	48.30	63.22
+SameProperName	69.11	72.81	48.56	63.49
+ModifierMismatch	69.11	72.81	48.58	63.50
+PropertyMismatch	69.20	72.89	48.67	63.59

Table 6.4: Ablation study on constraints. Performance on OntoNotes-5.0 data with predicted mentions.

6.6.4 Ablation Study of Constraints

Finally, we study the value of individual constraints by adding one constraint at a time into the coreference system starting with the simple L^3M model. Table 6.4 shows the results on Ontonotes dataset with predicted mentions. Overall, it is shown that each one of the constraints helps, and that using all the constraints improves the performance of the system by 1.29% in the AVG F1 score. In particular, most of this improvement (1.19%) is due to the must-link constraints (the first four constraints in the table.) The must-link constraints are more useful for L^3M as L^3M achieves higher precision than recall (e.g., the precision and recall of L^3M are 78.38% and 67.96%, respectively in B^3). As a result, the must-link constraints, which aim to improve the recall, do better when optimizing F1.

6.7 Conclusion

We presented a principled and straightforward framework for coreference resolution. We augment our model with domain knowledge-based constraints. We also presented a probabilistic generalization of this model that can take into account multiple possible coreference links. We proposed a fast stochastic gradient-based learning technique for our model. Our model, while operating at mention pair granularity, obtains state-of-the-art results on OntoNotes-5.0, and performs especially well on mention clusters containing named entity. We provided a detailed analysis of our experimental results.

Some efforts have been made (Haghighi and Klein, 2010; Rahman and Ng, 2011a,b) to consider models that capture higher order interactions, in particular, between mentions and previously

identified entities (that is, between mentions and clusters.) While such models are potentially more expressive, they are largely based on heuristics. As we discussed in Section 6.4, our PL^{3}M naturally allows us to consider mention-to-entity links.

Chapter 7

Conclusion and Discussion

In this thesis, we presented selective learning algorithms for training expressive models under a diverse set of scenarios. In all cases, we show that efficient learning can be achieved by carefully selecting and caching samples, structures or latent items. We analyzed the proposed algorithms and provided extensive experimental results. In this chapter, we conclude the thesis by discussing some directions for future research.

The selective block minimization framework proposed in Chapter 3 can be applied broadly. For example, it can be applied for solving L2-loss SVM, regularized least square problems, and logistic regression, by applying the corresponding dual solvers (Hsieh et al., 2008; Yu et al., 2010). Moreover, in Chapter 3 we considered sparse data stored in row format, such that we could split the data into blocks and access particular instances. When data is stored in a column format, splitting can only be done feature by feature. It would be interesting to investigate how to extend our method to this scenario.

The DEMI-DCD algorithm proposed in Chapter 4 opens up several directions for future research. Here, we have considered the case of a single learning thread that updates the models using the available active sets. A promising direction for future exploration is to study the possibility of using multiple learning threads to update the models. For some structured prediction problems, the output structure may be too complex for inference to be solved in a tractable fashion. For such cases, learning schemes that use approximate inference have been proposed (e.g., (Finley and Joachims, 2008; Meshi et al., 2010)). Incorporating approximate inference into our method is an interesting topic for future study.

There are several directions worth exploring to improve the AI-DCD algorithms proposed in Chapter 5, including a better implementation that could further help to reduce wall clock time, experimental study of additional structured prediction problems, and the study of two additional

amortization theorems (Theorems 6 and 7) and their impact on training structured prediction. These theorems require solving a linear programming problem for checking amortization. We observed that although these theorems reduce the number of inference calls, they do not lead to significant performance gain in running time, because invoking linear programming solvers costs additional overhead. A better implementation of these theorems has a great potential to further speed-up the training process. The margin-based amortized inference (e.g., Theorem 7) also provides better amortization. However, checking validity of the condition involves computing the structured margin for each cached solution, which involves solving an additional ILP problem. How to cache the solutions under such setting in an online fashion is a future research direction.

The L^3M model proposed in Chapter 6 can be broadly applied in other supervised clustering problems such as clustering posts in a forum or error reports by users (Samdani et al., 2014). Extending and applying L^3M to these real-world applications is a great further research topic. Moreover, the complexity of the inference algorithm described in Chapter 6 is quadratic in the number of items. Designing an efficient inference algorithm that scales linearly with the number of items is important for applications with a large number of items.

The idea of selecting informative components can be broadly applied in other learning scenarios. For example, this idea can be used to accelerate tensor decomposition methods for exploring relations in structured data (Chang et al., 2014). In a knowledge base, a fact can be represented as an entity-relation triple $(e_1; r; e_2)$ in which entities e_1 and e_2 have a relation r. These triples can be naturally encoded in a three-way tensor $X \in \{0,1\}^{n \times n \times m}$, such that $X_{i,j,k} = 1$ if and only if the triple $(e_i; r_k; e_j)$ is in the knowledge base, where n is the number of entities and m is the number of relations. To identify latent components in a tensor for collective learning, Nickel et al. (2011, 2012) proposed RESCAL, which is a tensor decomposition approach specifically designed for the multi-relational data described above. When factorizing a large knowledge base with 753,000 entities and 200 relations, the number of entries in the tensor can be extremely large (98 trillion entries), thus resulting in a scalability issue. In (Chang et al., 2014), we design a selective tensor decomposition algorithm by leveraging relational domain knowledge about entity type information. We exclude triples that do not satisfy the relational constraints (e.g., both arguments of the relation spouse-of need to be person entities) from the loss by selecting sub-matrices for each slice of the

tensor during training. Consequently, our method is significantly faster than previous approaches and is more accurate in discovering unseen triples. It would be useful to explore other learning algorithms that could benefit from the idea of selection.

References

A. Agarwal and J. Duchi. Distributed delayed stochastic optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *The Conference on Advances in Neural Information Processing Systems (NIPS)*, 2011.

A. Agarwal, O. Chapelle, M. Dudík, and J. Langford. A reliable effective terascale linear learning system. *Journal of Machine Learning Research*, 2014.

A. Bagga and B. Baldwin. Algorithms for scoring coreference chains. In *In The First International Conference on Language Resources and Evaluation Workshop on Linguistics Coreference*, 1998.

J. L. Balcázar, Y. Dai, and O. Watanabe. A random sampling technique for training support vector machines. In *ALT*, 2001.

M. Bansal and D. Klein. Coreference semantics from web features. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, July 2012.

N. Bansal, A. Blum, and S. Chawla. Correlation clustering. In *IEEE Symp. of Foundation of Computer Science*, 2002.

E. Bengtson and D. Roth. Understanding the value of features for coreference resolution. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, Oct 2008.

A. Björkelund and R. Farkas. Data-driven multilingual coreference resolution using resolver stacking. In *Joint Conference on EMNLP and CoNLL - Shared Task*, 2012.

A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, 2005.

L. Bottou and Y. LeCun. Large scale online learning. In *The Conference on Advances in Neural Information Processing Systems (NIPS)*, 2004.

L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

E. Chang, K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, and H. Cui. Parallelizing support vector machines on distributed computers. In *NIPS 21*, 2007a.

K.-W. Chang and D. Roth. Selective block minimization for faster convergence of limited memory large-scale linear models. In *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD)*, 2011.

K.-W. Chang, R. Samdani, A. Rozovskaya, N. Rizzolo, M. Sammons, and D. Roth. Inference protocols for coreference resolution. In *Proceedings of the Annual Conference on Computational Natural Language Learning (CoNLL)*, 2011.

K.-W. Chang, R. Samdani, A. Rozovskaya, M. Sammons, and D. Roth. Illinois-coref: The ui system in the conll-2012 shared task. In *Proc. of the Conference on Computational Natural Language Learning (CoNLL) Shared Task*, 2012a.

K.-W. Chang, R. Samdani, and D. Roth. A constrained latent variable model for coreference resolution. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, 2013a.

K.-W. Chang, V. Srikumar, and D. Roth. Multi-core structural SVM training. In *Proceedings of the European Conference on Machine Learning (ECML)*, 2013b.

K.-W. Chang, S. Upadhyay, G. Kundu, and D. Roth. Structural learning with amortized inference. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2015.

K.-W. Chang, S. Sundararajan, S. S. Keerthi, and S. Sellamanickam. Tractable semi-supervised learning of complex structured prediction models. In *Proceedings of the European Conference on Machine Learning (ECML)*, 2013c.

K.-W. Chang, W.-t. Yih, B. Yang, and C. Meek. Typed tensor decomposition of knowledge bases for relation extraction. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, 2014.

M. Chang and W. Yih. Dual coordinate descent algorithms for efficient large margin structural learning. *Transactions of the Association for Computational Linguistics*, 2013.

M. Chang, L. Ratinov, and D. Roth. Guiding semi-supervision with constraint-driven learning. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, Jun 2007b.

M. Chang, V. Srikumar, D. Goldwasser, and D. Roth. Structured output learning with indirect supervision. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2010.

M. Chang, L. Ratinov, and D. Roth. Structured learning with constrained conditional models. *Machine Learning*, 88(3):399–431, 6 2012b.

C. Chen and V. Ng. Combining the best of two worlds: A hybrid approach to multilingual coreference resolution. In *Joint Conference on EMNLP and CoNLL - Shared Task*, 2012.

C. Chu, S. K. Kim, Y. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. *The Conference on Advances in Neural Information Processing Systems (NIPS)*, 19:281, 2007.

J. Clarke and M. Lapata. Global inference for sentence compression: An integer linear programming approach. *Journal of Artificial Intelligence Research (JAIR)*, 31:399–429, 2008.

J. Clarke and M. Lapata. Constraint-based sentence compression: An integer programming approach. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, July 2006.

M. Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, 2002.

M. Collins and B. Roark. Incremental parsing with the perceptron algorithm. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2004.

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 2006.

K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. In *Computational Learing Theory*, 2000.

K. Crammer, M. Dredze, and A. Kulesza. Multi-class confidence weighted algorithms. In *EMNLP*, 2009.

A. Culotta, M. Wick, R. Hall, and A. McCallum. First-order probabilistic models for coreference resolution. In *Proceedings of the Annual Meeting of the North American Association of Computational Linguistics (NAACL)*, 2007.

O. Dekel, S. Shalev-Shwartz, and Y. Singer. The forgetron: A kernel-based perceptron on a budget. *SIAM Journal on Computing*, 37:1342–1372, January 2008.

P. Denis and J. Baldridge. Joint determination of anaphoricity and coreference resolution using integer programming. In *Proceedings of the Annual Meeting of the North American Association of Computational Linguistics (NAACL)*, 2007.

P. Denis and J. Baldridge. Specialized models and ranking for coreference resolution. In *EMNLP*, 2008.

Q. Do, D. Roth, M. Sammons, Y. Tu, and V. Vydiswaran. Robust, light-weight approaches to compute lexical similarity. Technical report, Computer Science Department, University of Illinois, 2009.

M. Dredze, K. Crammer, and F. Pereira. Confidence-weighted linear classification. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2008.

G. Durrett, D. Hall, and D. Klein. Decentralized entity-level modeling for coreference resolution. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, August 2013.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training support vector machines. *Journal of Machine Learning Research*, 6:1889–1918, 2005.

E. R. Fernandes, C. N. dos Santos, and R. L. Milidiú. Latent structure perceptron with feature induction for unrestricted coreference resolution. In *Joint Conference on EMNLP and CoNLL - Shared Task*, 2012.

T. Finley and T. Joachims. Supervised clustering with support vector machines. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2005.

T. Finley and T. Joachims. Training structural SVMs when exact inference is intractable. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2008.

Y. Freund, R. E. Schapire, Y. Singer, and M. K. Warmuth. Using and combining predictors that specialize. In *ACM Symp. of the Theory of Computing*, 1997.

K. Ganchev, J. Graça, J. Gillenwater, and B. Taskar. Posterior regularization for structured latent variable models. *Journal of Machine Learning Research*, 2010.

M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990. ISBN 0716710455.

K. Gimpel and N. A. Smith. Softmax-margin CRFs: Training log-linear models with cost functions. In *Proceedings of the Annual Meeting of the North American Association of Computational Linguistics (NAACL)*, 2010.

S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE Trans. on Knowl. and Data Eng.*, 2003.

Gurobi Optimization, Inc. Gurobi optimizer reference manual, 2012.

I. Guyon, D. Henderson, P. Albrecht, Y. LeCun, and J. Denker. Writer independent and writer adaptive neural network for on-line character recognition. *From Pixels to Feature III: Frontiers in Handwriting Recognition. Elsevier Science Publishers*, 1992.

A. Haghighi and D. Klein. Coreference resolution in a modular, entity-centered model. In *Proceedings of the Annual Meeting of the North American Association of Computational Linguistics (NAACL)*, June 2010.

P. Haider, U. Brefeld, and T. Scheffer. Supervised clustering of streaming data for email batch detection. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2007.

S. Har-Peled, D. Roth, and D. Zimak. Constraint classification for multiclass classification and ranking. In *The Conference on Advances in Neural Information Processing Systems (NIPS)*, 2003.

T. Hazan and R. Urtasun. A primal-dual message-passing algorithm for approximated large scale structured prediction. In *The Conference on Advances in Neural Information Processing Systems (NIPS)*, 2010.

D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. M. Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1:49–75, 2000.

M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, 1971.

C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2008. ISBN 978-1-60558-205-4.

L. Huang, S. Fayong, and Y. Guo. Structured perceptron with inexact search. In *Proceedings of the Annual Meeting of the North American Association of Computational Linguistics (NAACL)*, 2012.

H. Ji and R. Grishman. Knowledge base population: successful approaches and challenges. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2011.

T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA, 1999.

T. Joachims, T. Finley, and C.-N. Yu. Cutting-plane training of structural SVMs. *Machine Learning*, 77(1):27–59, 2009.

T. Joachims. Training linear SVMs in linear time. In *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD)*, 2006.

S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to platt's smo algorithm for svm classifier design. *Neural Computation*, 2001.

S. S. Keerthi, S. Sundararajan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. A sequential dual method for large scale multi-class linear SVMs. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.

R. Kindermann and J. L. Snell. *Markov random fields and their applications.* American Mathematical Society, 1980.

T. Koo, A. M. Rush, M. Collins, T. Jaakkola, and D. Sontag. Dual decomposition for parsing with non-projective head automata. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, 2010.

F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, pages 498–519, 2001.

A. Kulesza and F. Pereira. Structured learning with approximate inference. In *The Conference on Advances in Neural Information Processing Systems (NIPS)*, 2008.

G. Kundu, V. Srikumar, and D. Roth. Margin-based decomposed amortized inference. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 8 2013.

S. Lacoste-Julien, M. Jaggi, and P. P. Mark Schmidt. Block-coordinate Frank-Wolfe optimization for structural SVMs. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2013.

J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2001.

J. Langford, L. Li, and T. Zhang. Sparse online learning via truncated gradient. *Journal of Machine Learning Research*, 10:771–801, 2009a.

J. Langford, A. J. Smola, and M. Zinkevich. Slow learners are fast. In *The Conference on Advances in Neural Information Processing Systems (NIPS)*, 2009b.

E. Lassalle and P. Denis. Joint anaphoricity detection and coreference resolution with constrained latent structures. 2015.

H. Lee, Y. Peirsman, A. Chang, N. Chambers, M. Surdeanu, and D. Jurafsky. Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task. In *Proceedings of the CoNLL-2011 Shared Task*, 2011.

H. Lee, A. Chang, Y. Peirsman, N. Chambers, M. Surdeanu, and D. Jurafsky. Deterministic coreference resolution based on entity-centric, precision-ranked rules. *Computational Linguistics*, 39(4), 2013.

G. Loosli, S. Canu, and L. Bottou. Training invariant support vector machines using selective sampling. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*, pages 301–320. MIT Press, Cambridge, MA., 2007.

X. Luo. On coreference resolution performance metrics. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, 2005.

Z.-Q. Luo and P. Tseng. On the convergence of coordinate descent method for convex differentiable minimization. *Journal of Optimization Theory and Applications*, 72(1):7–35, 1992.

Z.-Q. Luo and P. Tseng. Error bounds and convergence analysis of feasible descent methods: a general approach. *Annals of Operations Research*, 46:157–178, 1993.

C. Ma, J. R. Doppa, J. W. Orr, P. Mannem, X. Fern, T. Dietterich, and P. Tadepalli. Prune-and-score: Learning for greedy coreference resolution. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, 2014.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*.

S. Martschat, J. Cai, S. Broscheit, E. Mújdricza-Maydt, and M. Strube. A multigraph model for coreference resolution. In *Joint Conference on EMNLP and CoNLL - Shared Task*, July 2012.

A. Mccallum and B. Wellner. Toward conditional models of identity uncertainty with application to proper noun coreference. In *The Conference on Advances in Neural Information Processing Systems (NIPS)*, 2003.

R. McDonald, K. Hall, and G. Mann. Distributed training strategies for the structured Perceptron. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, June 2010a.

R. McDonald, K. Hall, and G. Mann. Distributed training strategies for the structured Perceptron. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, June 2010b.

O. Meshi, D. Sontag, T. Jaakkola, and A. Globerson. Learning efficiently with approximate inference via dual losses. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2010.

R. Mihalcea and A. Csomai. Wikify!: linking documents to encyclopedic knowledge. In *Proceedings of ACM Conference on Information and Knowledge Management (CIKM)*, 2007.

M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, 1969.

A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *The Conference on Advances in Neural Information Processing Systems (NIPS)*, 2001.

V. Ng. Supervised noun phrase coreference research: the first fifteen years. In *ACL*, 2010.

V. Ng and C. Cardie. Improving machine learning approaches to coreference resolution. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2002.

V. Ng. Supervised ranking for pronoun resolution: Some recent improvements. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2005.

M. Nickel, V. Tresp, and H.-P. Kriegel. A three-way model for collective learning on multi-relational data. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2011.

M. Nickel, V. Tresp, and H.-P. Kriegel. Factorizing YAGO: scalable machine learning for linked data. In *The International World Wide Web Conference*, 2012.

NIST. The ACE evaluation plan., 2004.

D. Pascal and J. Baldridge. Global joint models for coreference resolution and named entity classification. In *Procesamiento del Lenguaje Natural*, 2009.

J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, San Mateo (Calif.), 1988.

F. Pérez-Cruz, A. R. Figueiras-Vidal, and A. Artés-Rodríguez. Double chunking for solving SVMs for very large datasets. In *Proceedings of Learning 2004, Spain*, 2004.

J. C. Platt. Fast training of support vector machines using sequential minimal optimization. *Advances in kernel methods: support vector learning*, 1999.

P. Pletscher, C. S. Ong, and J. M. Buhmann. Entropy and margin maximization for structured output learning. In *Proceedings of the European Conference on Machine Learning (ECML)*, 2010.

S. Pradhan, L. Ramshaw, M. Marcus, M. Palmer, R. Weischedel, and N. Xue. Conll-2011 shared task: Modeling unrestricted coreference in ontonotes. In *Proceedings of the Annual Conference on Computational Natural Language Learning (CoNLL)*, 2011.

S. Pradhan, A. Moschitti, N. Xue, O. Uryupina, and Y. Zhang. CoNLL-2012 shared task: Modeling multilingual unrestricted coreference in OntoNotes. In *Proceedings of the Annual Conference on Computational Natural Language Learning (CoNLL)*, 2012.

V. Punyakanok, D. Roth, and W. Yih. The necessity of syntactic parsing for semantic role labeling. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2005a.

V. Punyakanok, D. Roth, W. Yih, and D. Zimak. Learning and inference over constrained output. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2005b.

A. Quattoni, S. Wang, L.-P. Morency, M. Collins, and T. Darrell. Hidden conditional random fields. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(10):1848–1852, 2007. ISSN 0162-8828.

K. Raghunathan, H. Lee, S. Rangarajan, N. Chambers, M. Surdeanu, D. Jurafsky, and C. Manning. A multi-pass sieve for coreference resolution. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, 2010.

A. Rahman and V. Ng. Ensemble-based coreference resolution. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2011a.

A. Rahman and V. Ng. Narrowing the modeling gap: a cluster-ranking approach to coreference resolution. *Journal of AI Research*, 2011b.

A. Rahman and V. Ng. Coreference resolution with world knowledge. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2011c.

L. Ratinov and D. Roth. Learning-based multi-sieve co-reference resolution with knowledge. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, 2012.

S. Riedel and J. Clarke. Incremental integer linear programming for non-projective dependency parsing. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, 2006.

D. Roth and W. Yih. A linear programming formulation for global inference in natural language tasks. In H. T. Ng and E. Riloff, editors, *Proceedings of the Annual Conference on Computational Natural Language Learning (CoNLL)*, 2004.

D. Roth and W. Yih. Integer linear programming inference for conditional random fields. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2005.

D. Roth and W. Yih. Global inference for entity and relation identification via a linear programming formulation. In L. Getoor and B. Taskar, editors, *Introduction to Statistical Relational Learning*, 2007.

A. Rozovskaya and D. Roth. Training paradigms for correcting errors in grammar and usage. In *Proceedings of the Annual Meeting of the North American Association of Computational Linguistics (NAACL)*, Jun 2010.

A. Rush and M. Collins. A tutorial on dual decomposition and lagrangian relaxation for inference in natural language processing. *Journal of Machine Learning Research*, 2002.

A. M. Rush, D. Sontag, M. Collins, and T. Jaakkola. On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, 2010.

R. Samdani and D. Roth. Efficient decomposed learning for structured prediction. In *Proceedings of the International Conference on Machine Learning (ICML)*, 6 2012.

R. Samdani, M. Chang, and D. Roth. Unified expectation maximization. In *Proceedings of the Annual Meeting of the North American Association of Computational Linguistics (NAACL)*, 6 2012a.

R. Samdani, M. Chang, and D. Roth. A framework for tuning posterior entropy in unsupervised learning, 6 2012b.

R. Samdani, K.-W. Chang, and D. Roth. A discriminative latent variable model for online clustering. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2014.

K. Scheinberg. An efficient implementation of an active set method for svms. *Journal of Machine Learning Research*, 7:2237–2257, 2006.

A. Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, Inc., 1986.

S. Shalev-Shwartz and N. Srebro. SVM optimization: inverse dependence on training set size. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2008.

S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: primal estimated sub-gradient solver for SVM. In Z. Ghahramani, editor, *Proceedings of the International Conference on Machine Learning (ICML)*, 2007.

S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss. *Journal of Machine Learning Research*, 14(1):567–599, 2013.

D. Shen, Q. Yang, J.-T. Sun, and Z. Chen. Thread detection in dynamic text message streams. In *Proceedings of International Conference on Research and Development in Information Retrieval, SIGIR*, 2006.

S. K. Shevade, B. P., S. Sundararajan, and S. S. Keerthi. A sequential dual method for structural SVMs. In *SDM*, 2011.

S. Sonnenburg and V. Franc. COFFIN: a computational framework for linear svms. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2010.

D. Sontag. *Approximate Inference in Graphical Models using LP Relaxations.* PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2010.

W. M. Soon, H. T. Ng, and D. C. Y. Lim. A machine learning approach to coreference resolution of noun phrases. *Comput. Linguist.*, 2001.

V. Srikumar, G. Kundu, and D. Roth. On amortizing inference cost for structured prediction. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, 7 2012.

V. Stoyanov and J. Eisner. Easy-first coreference resolution. In *Proceedings the International Conference on Computational Linguistics (COLING)*, 2012.

V. Stoyanov, N. Gilbert, C. Cardie, and E. Riloff. Conundrums in noun phrase coreference resolution: making sense of the state-of-the-art. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2009.

C. Sutton and A. McCallum. Piecewise pseudolikelihood for efficient training of conditional random fields. In Z. Ghahramani, editor, *Proceedings of the International Conference on Machine Learning (ICML)*, 2007.

B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In *The Conference on Advances in Neural Information Processing Systems (NIPS)*, 2004.

B. Taskar, V. Chatalbashev, D. Koller, and C. Guestrin. Learning structured prediction models: a large margin approach. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2005.

S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, 2:45–66, 2002.

I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, September 2005.

M. Vilain, J. Burger, J. Aberdeen, D. Connolly, and L. Hirschman. A model-theoretic coreference scoring scheme. In *Proceedings of the 6th conference on Message understanding*, 1995.

S. V. N. Vishwanathan, A. J. Smola, and M. N. Murty. Simplesvm. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2003.

P.-W. Wang and C.-J. Lin. Iteration complexity of feasible descent methods for convex optimization. *Technical Report, National Taiwan University*, 2013.

K. Woodsend and J. Gondzio. Hybrid mpi/openmp parallel linear support vector machine training. *Journal of Machine Learning Research*, 10:1937–1953, 2009.

J. S. Yedidia, W. T. Freeman, and Y. Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005.

C. Yu and T. Joachims. Learning structural svms with latent variables. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2009.

H.-F. Yu, C.-J. Hsieh, K.-W. Chang, and C.-J. Lin. Large linear classification when data cannot fit in memory. In *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD)*, 2010.

H. Yu, J. Yang, and J. Han. Classifying large data sets using SVMs with hierarchical clusters. In *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD)*, 2003.

A. L. Yuille and A. Rangarajan. The concave-convex procedure. *Neural Computation*, 15(4), 2003.

M. Zinkevich, M. Weimer, A. Smola, and L. Li. Parallelized stochastic gradient descent. In *The Conference on Advances in Neural Information Processing Systems (NIPS)*, 2010.

Appendix A

Proofs

A.1 Proof of Theorem 1

We follow the proof in (Yu et al., 2010, Theorem 1). Theorem 2.1 of Luo and Tseng (1992) analyzes the convergence of coordinate descent methods. The theorem requires several conditions on (2.4) and an almost cyclic update rule. (Hsieh et al., 2008, Theorem 1)) have proved that Eq. (2.4) satisfies the required conditions. To satisfy the second condition, we need to prove that, in the coordinate decent method, there exists an integer T such that every variable is updated at least once between the rth iteration and the $(r+T)$th iteration, for all r. From Algorithm 2, for each α_i, there exists an index j such that $\alpha_i \in B_j$ and $B_j \subset W^{t,j}$. Therefore, the two assumptions imply α_i is updated at least once and the number of updates is bounded at each outer iteration. Therefore, Algorithm 2 enjoys both global and linear convergence.

A.2 Proof of Theorem 2

Let $\boldsymbol{\alpha}^*$ be the optimal solution of (2.5), and $\boldsymbol{\alpha}'$ is the optimal solution of the following optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha} \\ &\text{subject to } \boldsymbol{\alpha}_{\bar{V}} = 0, 0 \le \alpha_i \le C, i = 1, \ldots, l. \end{aligned} \tag{A.1}$$

The solution space of (2.5) is larger than (A.1). Therefore, $\boldsymbol{\alpha}'$ is a feasible solution of (2.5). Since $\boldsymbol{\alpha}^*$ is an optimum of (2.5),

$$f(\boldsymbol{\alpha}^*) \le f(\boldsymbol{\alpha}'). \tag{A.2}$$

Since $\boldsymbol{\alpha}^*$ satisfies the constraint $\boldsymbol{\alpha}_{\bar{V}} = 0$, $\boldsymbol{\alpha}^*$ is a feasible solution of (A.1). As $\boldsymbol{\alpha}'$ is an optimum of (A.1),

$$f(\boldsymbol{\alpha}') \leq f(\boldsymbol{\alpha}^*). \tag{A.3}$$

Combining (A.2) and (A.3), we obtain

$$f(\boldsymbol{\alpha}') = f(\boldsymbol{\alpha}^*). \tag{A.4}$$

A.3 Proof of Theorem 8

For notation simplicity, we slightly abuse the notation and use $z^{\mathbf{p}}_{j,p}, z^{\mathbf{p}}_{j,q}, z^{\mathbf{q}}_{j,p}, z^{\mathbf{q}}_{j,q}$ to represent $z^{\mathbf{p}}_{j,y^p_j}, z^{\mathbf{p}}_{j,y^q_j}, z^{\mathbf{q}}_{j,y^p_j}, z^{\mathbf{q}}_{j,y^q_j}$, respectively. Note that the feasible set of problems $\mathbf{p}$ and $\mathbf{q}$ are the same. Therefore, $z^{\mathbf{p}}$ and $z^{\mathbf{q}}$ are both feasible solutions to problems $\mathbf{p}$ and $\mathbf{q}$. Because $z^{\mathbf{p}}$ is the optimal solution to $\mathbf{p}$, we have

$$f^{\mathbf{p}}(\boldsymbol{z}^{\mathbf{p}}) = \sum_j c^{\mathbf{p}}_{j,p} \geq \sum_j c^{\mathbf{p}}_{j,q} = f^{\mathbf{p}}(\boldsymbol{z}^{\mathbf{q}}).$$

Moreover, if $y^{\mathbf{p}}_j = y^{\mathbf{q}}_j$, we have $c^{\mathbf{p}}_{j,p} = c^{\mathbf{p}}_{j,q}$. Therefore,

$$\sum_{j: y^{\mathbf{p}}_j != y^{\mathbf{q}}_j} c^{\mathbf{p}}_{j,p} \geq \sum_{j: y^{\mathbf{p}}_j != y^{\mathbf{q}}_j} c^{\mathbf{p}}_{j,q}$$

From condition (5.6), and $z^{\mathbf{p}}_{j,p} = 1$, we have

$$c^{\mathbf{p}}_{j,p} - c^{\mathbf{q}}_{j,p} \leq \epsilon |c^{\mathbf{q}}_{j,p}|, \quad \forall j$$

When $y^j_q \neq y^j_p$, we have $z^{\mathbf{p}}_{j,q} = 0$ and

$$c^{\mathbf{q}}_{j,q} - c^{\mathbf{p}}_{j,q} \leq \epsilon |c^{\mathbf{q}}_{j,q}|$$

Then, we are ready to derive the theorem:

$$
\begin{aligned}
f^{\mathbf{q}}(\boldsymbol{z}^{\mathbf{q}}) - f^{\mathbf{q}}(\boldsymbol{z}^{\mathbf{p}}) &= \sum_j c^{\mathbf{q}}_{j,q} - c^{\mathbf{q}}_{j,p} \\
&= \sum_{j:y^p_j \neq y^q_j} c^{\mathbf{q}}_{j,q} - c^{\mathbf{q}}_{j,p} \\
&\leq \sum_{j:y^p_j \neq y^q_j} \epsilon |c^{\mathbf{q}}_{j,q}| + c^{\mathbf{p}}_{j,q} - c^{\mathbf{q}}_{j,p} \\
&\leq \sum_{j:y^p_j \neq y^q_j} \epsilon |c^{\mathbf{q}}_{j,q}| + c^{\mathbf{p}}_{j,p} - c^{\mathbf{q}}_{j,p} \\
&\leq \sum_{j:y^p_j \neq y^q_j} \epsilon |c^{\mathbf{q}}_{j,q}| + \epsilon |c^{\mathbf{q}}_{j,p}| \\
&\leq \epsilon \sum_j (|c^{\mathbf{q}}_{j,q}| + |c^{\mathbf{q}}_{j,p}|) \\
&\leq M \epsilon f^{\mathbf{q}}(\boldsymbol{z}^{\mathbf{p}}).
\end{aligned}
$$

Rearranging the terms proves the theorem.